

Programação Avançada: Lista de Exercícios (Ponteiros)

Nome: Caio Mendonça de
Andrade

Matricula: 20230094570

Listas enviadas fora desse padrão serão penalizadas em 2 pontos.

Questões

1. Seja o seguinte trecho de programa:

```
int i=3,j=5;  
int *p, *q;  
p = &i;  
q = &j;
```

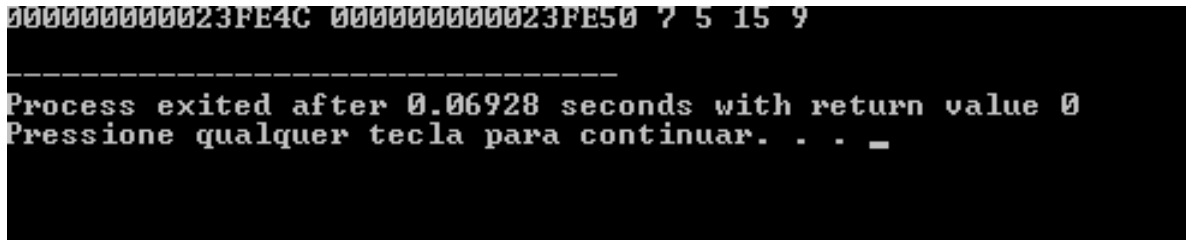
Determine o valor das seguintes expressões, justificando o porquê de cada resultado:

```
Resultado da expressão p == &i: 1  
Resultado da expressão *p - *q: -2  
Resultado da expressão **&p: 3  
Resultado da expressão 3 - *p/(*q) + 7: 10  
-----  
Process exited after 0.03109 seconds with return value 0  
Pressione qualquer tecla para continuar. . . _
```

- `p == &i`; verifica se o valor armazenado em `p` é igual ao endereço de `i`. Como `p` foi definido como o endereço de `i`, esta expressão será verdadeira (1 ou true) porque `p` e `&i` apontam para o mesmo local na memória.
- `*p - *q`; `*p` e `*q` desreferenciam os ponteiros `p` e `q`, respectivamente, ou seja, eles retornam os valores armazenados nos endereços de memória para onde `p` e `q` apontam. Então, `*p` é 3 e `*q` é 5. Portanto, a expressão se torna `3 - 5`, resultando em -2.
- `**&p`; `&p` retorna o endereço de `p`, que é o endereço de memória onde `p` está armazenado. `*` então desreferencia esse endereço, o que significa que estamos acessando o valor armazenado nesse endereço, que é o próprio endereço de `i`. Em seguida, o segundo `*` desreferencia esse endereço, então o valor final será o valor de `i`, que é 3.
- `3 - *p/(*q) + 7`; Primeiro, calculamos `*p` que é 3 e `*q` que é 5. Então, a expressão se torna `3 - 3/5 + 7`. Resolvendo a divisão primeiro, `3/5` resulta em 0 (pois é uma divisão entre inteiros), então a expressão se torna `3 - 0 + 7`, que é igual a 10.

2. Mostre o que será impresso por programa supondo que a variável `i` ocupa o endereço 4094 na memória e que nessa arquitetura os inteiros possuem 2 bytes de tamanho.

```
main(){
    int i=5, *p;
    p = &i;
    printf("%p %p %d %d %d %d\n", p, p+1, *p+2, **&p, 3**p, **&p+4);
```



```
000000000023FE4C 000000000023FE50 7 5 15 9
-----
Process exited after 0.06928 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

3. Se `i` e `j` são variáveis inteiras e `p` e `q` ponteiros para `int`, quais das seguintes expressões de atribuição são ilegais? Justifique.

```
p = &i;
*q = &j;
p = *&i;
i = (*&)j;
i = *&j;
i = **&j;
q = *p;
i = (*p)++ + *q;
```

`p = &i;` Esta expressão é legal. A variável `p`, que é um ponteiro para inteiro, está sendo atribuída com o endereço de `i`, que é uma variável inteira.

`*q = &j;` Esta expressão é ilegal. `*q` é um valor inteiro, enquanto `&j` é um endereço de memória (ponteiro para inteiro). Você não pode atribuir um endereço de memória a uma variável inteira.

`p = *&i;` Esta expressão é legal. `&i` retorna o endereço de `i`, `*` desreferencia esse endereço, e `&` pega o endereço dessa desreferência novamente. Portanto, `&*i` é equivalente a `&i`, que é um endereço de `i`, e pode ser atribuído a `p`.

`i = (*&)j;` Esta expressão é ilegal. `(*&)j` é uma tentativa de desreferenciar o endereço de `j`, o que não é permitido. O operador `&` precisa de uma variável lvalue (que pode ser uma expressão cujo valor pode ser usado como o lado esquerdo de uma atribuição), mas `&j` retorna um valor rvalue (um valor que não pode ser atribuído).

`i = *&j;` Esta expressão é legal. `*&j` é equivalente a `j`, pois `*` desreferencia o endereço de `j`, e `&` pega o endereço da desreferência. Portanto, `*&j` é apenas `j`, que é um valor inteiro que pode ser atribuído a `i`.

`i = **&j;` Esta expressão é legal. `**&j` é equivalente a `j`, pois `*&` desreferencia o endereço de `j` duas vezes, resultando no valor original de `j`, que é um valor inteiro que pode ser atribuído a `i`.

`q = *p;` Esta expressão é ilegal. `*p` retorna o valor apontado por `p`, que é um valor inteiro, enquanto `q` é um ponteiro para inteiro. Você não pode atribuir um valor inteiro a um ponteiro.

`i = (*p)++ + *q;` Esta expressão é ilegal. `(*p)++` tenta incrementar o valor apontado por `p`, o que é permitido. No entanto, `*q` tenta acessar o valor apontado por `q`, mas `q` não foi inicializado corretamente com um endereço válido.

4. Determine o que será mostrado pelo seguinte programa (compile-o, execute-o e verifique se foram obtidas as respostas esperadas, justificando o porque de cada uma).

```
#include <stdio.h>

int main() {
    int    valor;
    int    *p1;
    float  temp;
    float  *p2;
    char   aux;
    char   *nome = "Ponteiros";
    char   *p3;
    int    idade;
    int    vetor[3];
    int    *p4;
    int    *p5;

    /* (a) */
    valor = 10;
    p1 = &valor;
    *p1 = 20;
    printf("%d \n", valor);

    /* (b) */
    temp = 26.5;
    p2 = &temp;
    *p2 = 29.0;
    printf("%.1f \n", temp);

    /* (c) */
    p3 = &nome[0];
    aux = *p3;
    printf("%c \n", aux);
}
```



```
20
29.0
P
-----
Process exited after 0.09402 seconds with return value 3
Pressione qualquer tecla para continuar. . . _
```

```

/* (d) */
p3 = &nome[4];
aux = *p3;
printf("%c \n", aux);

/* (e) */
p3 = nome;
printf("%c \n", *p3);

/* (f) */
p3 = p3 + 4;
printf("%c \n", *p3);

/* (g) */
p3--;
printf("%c \n", *p3);

/* (h) */
vetor[0] = 31;
vetor[1] = 45;
vetor[2] = 27;
p4 = vetor;
idade = *p4;
printf("%d \n", idade);

/* (i) */
p5 = p4 + 1;
idade = *p5;
printf("%d \n", idade);

/* (j) */
p4 = p5 + 1;
idade = *p4;
printf("%d \n", idade);

/* (l) */
p4 = p4 - 2;
idade = *p4;
printf("%d \n", idade);

/* (m) */
p5 = &vetor[2] - 1;
printf("%d \n", *p5);

/* (n) */
p5++;
printf("%d \n", *p5);
return(0);
}

```

5. Determine o que será mostrado pelo seguinte programa (compile-o, execute-o e explique se foram obtidas as respostas esperadas).

```
int main(void){
    float vet[5] = {1.1,2.2,3.3,4.4,5.5};
    float *f;
    int i;
    f = vet;
    printf("contador/valor/valor/endereco/endereco\n");
    for(i = 0 ; i <= 4 ; i++){
        printf("i = %d",i);
        printf(" vet[%d] = %.1f",i, vet[i]);
        printf(" *(f + %d) = %.1f",i, *(f+i));
        printf(" &vet[%d] = %X",i, &vet[i]);
        printf(" (f + %d) = %X",i, f+i);
        printf("\n");
    }
}
```

6. Assumindo que `pulo[]` é um vetor do tipo `int`, quais das seguintes expressões referenciam o valor do terceiro elemento do vetor?

- `*(pulo + 2);`
- `*(pulo + 4);`
- `pulo + 4;`
- `pulo + 2;`

7. Considerando a declaração

```
int mat[4], *p, x;
```

quais das seguintes expressões são válidas? Justifique.

```
p = mat + 1;
p = mat;
p = mat;
x = (*mat);
```

`p = mat;` Esta expressão é válida. `mat` é o nome do array, que pode ser usado como um ponteiro para o primeiro elemento do array. Portanto, `p` pode receber o endereço do primeiro elemento de `mat`.

`x = (*mat);` Esta expressão é válida. `*mat` desreferencia o primeiro elemento do array `mat` e atribui esse valor à variável `x`. Note que o resultado será o valor do primeiro elemento do array.

`p = mat + 1;` Esta expressão é válida. `mat` é um array de inteiros, então `mat + 1` resultará no endereço do segundo elemento do array. Como `p` é um ponteiro para inteiro, ele pode receber este endereço.

8. O que fazem os seguintes programas em C?

```
int main(){
    int vet[] = {4, 9, 13};
    int i;
    for(i=0;i<3;i++){
        printf("%d ", *(vet+i));
    }
}
```

```
4 9 13
-----
Process exited after 0.01448 seconds with return value 3
Pressione qualquer tecla para continuar. . . _
}
```

```
int main(){
    int vet[] = {4, 9, 13};
    int i;
    for(i=0;i<3;i++){
        printf("%X ", vet+i);
    }
}
```

```
23FE40 23FE44 23FE48
-----
Process exited after 0.01438 seconds with return value 7
Pressione qualquer tecla para continuar. . .
```

9. Qual será a saída do seguinte programa?

```
#include <stdio.h>
struct teste{
    int x = 3;
    char nome[] = "jose";
};
main(){
    struct teste *s;
    printf("%d", s->x);
    printf("%s", s->name);
}
```

10. Qual será a saída do seguinte programa

```
#include <stdio.h>
void main(){
    int const *x = 3;
    printf("%d", ++(*x));
}
```

11. Seja x um vetor de 4 elementos, declarado da forma TIPO x[4]. Suponha que depois da declaração, x esteja armazenado no endereço de memória 4092 (ou seja, o endereço de x[0]). Suponha também que na máquina seja usada uma variável do tipo char ocupa 1 byte, do tipo int ocupa 2 bytes, do tipo float ocupa 4 bytes e do tipo double ocupa 8 bytes. Quais serão os valores de x+1, x+2 e x+3 se:

- x for declarado como char?
- x for declarado como int?
- x for declarado como float?
- x for declarado como double?

Implemente um programa de computador para testar estas suposições e compare as respostas oferecidas pelo programa com as respostas que você idealizou.

12. Suponha que as seguintes declarações tenham sido realizadas:

```
float aloha[10], coisas[10][5], *pf, value = 2.2;
int i=3;
```

Identifique quais dos seguintes comandos é válido ou inválido:

```
aloha[2] = value;
scanf("%f", &aloha);
aloha = "value";
printf("%f", aloha);
coisas[4][4] = aloha[3];
coisas[5] = aloha;
pf = value;
pf = aloha;
```

aloha[2] = value; Válido. Este comando atribui o valor da variável value ao terceiro elemento do array aloha.

scanf("%f", &aloha); Inválido. O formato %f é para ler um float, mas &aloha é um ponteiro para o array aloha, não é um ponteiro para um float. O comando correto seria scanf("%f", &aloha[0]); para ler um float e armazená-lo no primeiro elemento do array aloha.

aloha = "value"; Inválido. Isso tenta atribuir uma string à variável aloha, que é um array de floats. Você não pode atribuir uma string diretamente a um array em C.

printf("%f", aloha); Inválido. Este comando tenta imprimir o array aloha como se fosse um float. Você precisa especificar um índice para acessar um elemento específico do array aloha.

`coisas[4][4] = aloha[3];` **Válido.** Este comando atribui o valor do quarto elemento do array `aloha` ao elemento `[4][4]` do array `coisas`.

`coisas[5] = aloha;` **Inválido.** Isso tenta atribuir o array `aloha` ao quinto elemento do array `coisas`, mas `coisas[5]` é uma linha (ou seja, um array de floats) e `aloha` é um array de floats. Você não pode atribuir um array diretamente a outro em C.

`pf = value;` **Inválido.** `pf` é um ponteiro para float e `value` é um float. Você não pode atribuir um float a um ponteiro sem usar o operador de endereço `&`.

`pf = aloha;` **Válido.** Isso atribui o endereço do primeiro elemento do array `aloha` ao ponteiro `pf`. Como `pf` é um ponteiro para float, isso é válido em C.

13. O que é memory leak? Procure 3 exemplos de programas em C que apresentem memory leak e explique o que acontece em cada um deles.
14. O que é um ponteiro para uma função? Pesquise na Internet referências sobre o assunto e escreva um pequeno programa exemplificando o uso deste recurso. Explique seu programa, comentando cada uma das linhas de código.
15. Implemente em linguagem C uma função em um programa de computador que leia `n` valores do tipo float e os apresente em ordem crescente. Utilize alocação dinâmica de memória para realizar a tarefa.
16. Reimplemente o programa da questão anterior utilizando a função `qsort()` do C. Comente o seu código, explicando o que faz cada uma das linhas.
17. Utilize a ideia do ponteiro para função pela função `qsort()` para implementar sua própria função de ordenação, mas que seja capaz de ordenar apenas inteiros do tipo `int`. Para isso, sua função deverá receber, entre outros argumentos, um ponteiro para a função de comparação que determinará como os elementos do array serão ordenados.
18. Procure na internet mecanismos que possibilitem medir tempos de execução de rotinas computacionais. Geralmente, estas medidas são realizadas com o auxílio de funções em C que lêem a hora no sistema (sistemas Unix e Windows geralmente usam funções diferentes). Utilizando os conhecimentos que você obteve com sua pesquisa, meça os tempos de execução das implementações que você criou para os dois problemas de ordenação anteriores, considerando apenas arrays de elementos tipo `int` e compare os resultados obtidos. O que se conclui nesse caso?
19. Escreva uma função em linguagem C que escreva em um vetor a soma dos elementos correspondentes de outros dois vetores (os tamanhos dos vetores devem ser fornecidos pelo usuário). Por exemplo, se o primeiro vetor contiver os elementos 1, 3, 0 e -2, e o segundo vetor contiver os elementos 3, 5, -3 e 1, o vetor de soma terá valores resultantes iguais a 4, 8, -3 e -1. A função deve receber 4 argumentos: os nomes dos três vetores e o número de elementos presentes em cada vetor. Exemplo:

```
soma_vetores(vet1, vet2, resultado, 4);
```

```
#include <stdio.h>
```

```
void soma_vetores(int i, int vet1[], int vet2[], int resultado[], int tamanho) {  
    for (i = 0; i < tamanho; i++) {  
        resultado[i] = vet1[i] + vet2[i];  
    }  
}
```

```
int main() {  
    int tamanho;
```

```
    printf("Digite o tamanho dos vetores: ");  
    scanf("%d", &tamanho);
```

```
    int i, vet1[tamanho], vet2[tamanho], resultado[tamanho];
```

```
    printf("Digite os elementos do segundo vetor:\n");  
    for (i = 0; i < tamanho; i++) {  
        scanf("%d", &vet2[i]);  
    }
```

```
    soma_vetores(i, vet1, vet2, resultado, tamanho);
```

```
    printf("Vetor de soma:\n");  
    for (i = 0; i < tamanho; i++) {  
        printf("%d ", resultado[i]);  
    }  
    printf("\n");
```

```
    return 0;
```

```
}
```

```
    printf("Digite os elementos do primeiro vetor:\n");  
    for (i = 0; i < tamanho; i++) {  
        scanf("%d", &vet1[i]);
```


20. Crie uma função capaz de realizar multiplicação matricial da forma $C = A \times B$. A função deve receber 6 argumentos: os ponteiros para as matrizes A, B e C, o número de linhas e colunas de A e o número de colunas de B (assuma que o número de coluna de A é igual ao número de linhas de B). O resultado da multiplicação deve ficar armazenado em C. Crie um programa para testar sua implementação, capaz de utilizar a função de multiplicação e imprimir as três matrizes. A função criada para multiplicação não deve realizar nenhum tipo de saída de dados no terminal. Exemplo: para multiplicar duas matrizes (A e B) de dimensões 2x3 e 3x4, respectivamente (o resultado deve ficar armazenado em C).
21. (ENADE, 2023) Memory leak, ou vazamento de memória, é um problema que ocorre em sistemas computacionais quando uma parte da memória, alocada para uma determinada operação, não é liberada quando se torna desnecessária. Na linguagem C, esse tipo de problema é quase sempre relacionado ao uso incorreto das funções `malloc()` e `free()`. Esse erro de programação pode levar a falhas no sistema se a memória for completamente consumida. Um dos trechos abaixo apresenta um vazamento de memória. Identifique-o e justifique sua resposta.

```
A void f( ){  
    void *s;  
    s = malloc(50);  
    free(s);  
}
```

Neste trecho, não há vazamento de memória, pois a memória alocada dinamicamente através de `malloc()` é liberada corretamente usando `free()` antes da função `f()` retornar.

```
B int f( ){  
    float *a;  
    return 0;  
}
```

Neste trecho, não há alocação de memória dinâmica, então não há possibilidade de vazamento de memória.

```
C int f(char *data){  
    void *s;  
    s = malloc(50);  
    int size = strlen(data);  
    if (size > 50)  
        return(-1);  
    free(s);  
    return 0;  
}
```

Neste trecho, pode haver um vazamento de memória se a condição `size > 50` for verdadeira. Se isso acontecer, a memória alocada para `s` não será liberada antes da função retornar, causando um vazamento de memória.

```
D int *f(int n){  
    int *num = malloc(sizeof(int)*n);  
    return num;  
}
```

```
int main(void){
    int *num;
    num = f(10);
    free(num);
    return 0;
}
```

Neste trecho, há um vazamento de memória. Embora a memória alocada para num seja liberada no main() usando free(num), a função f() aloca memória dinamicamente para num e retorna o ponteiro para ela. No entanto, após a função f() ser chamada e num receber o ponteiro retornado, não há uma chamada correspondente a free() para liberar essa memória alocada. Isso resulta em um vazamento de memória.

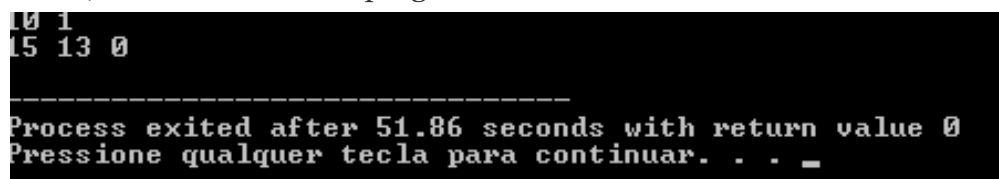
```
E void f(int n){
    char *m = malloc(10);
    char *n = malloc(10);
    free(m);
    m = n;
    free(m);
    free(n);
}
```

Neste trecho, não há vazamento de memória. A memória alocada para m é liberada corretamente usando free(m), antes de m receber o valor de n. Então, quando m é atribuído a n, ambos os ponteiros estão apontando para a mesma região de memória. Portanto, chamar free(m) libera essa região de memória, e chamar free(n) libera a mesma região de memória novamente. Isso é seguro em C, não causando vazamento de memória.

22. Na programação de sistemas embarcados, algumas posições de memória servem para diferentes propósitos, não apenas para armazenar valores. Em algumas dessas memórias, cada um dos bits possui um significado diferente, sendo necessário manipulá-los individualmente ou em pequenos grupos. Por isso, o conhecimento da álgebra booliana, bem como dos operadores tilizados para realizar operações binárias nas linguagens de programação, é essencial para o desenvolvimento desse tipo de sistema. A partir dessas informações, observe o código apresentado a seguir, escrito na linguagem C, que faz uso de operações binárias sobre variáveis inteiras.

```
#include <stdio.h>
int main(){
    int a, b;
    int x, y, z;
    scanf("%d %d", &a, &b);
    x = a; y = b; z = a + b;
    while (a) {
        x = x | b;
        y = y ^ a;
        z = z & (a+b);
        a = a >> 1;
        b = b << 1;
    }
    printf("%d %d %d\n", x, y, z);
    return 0;
}
```

Após a chamada desse programa, caso o usuário entre com os valores 10 e 1, nessa ordem, qual será, exatamente, o valor da saída do programa?



```
10 1
15 13 0
-----
Process exited after 51.86 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

23. (ENADE, 2021) Observe o código abaixo escrito na linguagem C.

```
#include <stdio.h>
#define TAM 10
int funcao1(int vetor[], int v){
    int i;
    for (i = 0; i < TAM; i++){
        if (vetor[i] == v)
            return i;
    }
    return -1;
}
int funcao2(int vetor[], int v, int i, int f){
    int m = (i + f) / 2;
    if (v == vetor[m])
        return m;
    if (i >= f)
        return -1;
    if (v > vetor[m])
        return funcao2(vetor, v, m+1, f);
    else
        return funcao2(vetor, v, i, m-1);
}
int main(){
    int vetor[TAM] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    printf("%d - %d", funcao1(vetor, 15), funcao2(vetor, 15, 0, TAM-1));
    return 0;
}
```

A respeito das funções implementadas, avalie as afirmações a seguir.

I. O resultado da impressão na linha 24 é: 7 - 7.

Isso está correto. A função `funcao1` retorna o índice do elemento no vetor `vetor` que é igual a 15, que é o valor passado como argumento. Como 15 está na sétima posição do vetor (índice 6, contando a partir de 0), ambas as funções retornam 7, que é a sétima posição do vetor. Portanto, a saída é 7 - 7.

II. A função `funcao1`, no pior caso, é uma estratégia mais rápida do que a `funcao2`.

Isso está incorreto. A função `funcao2` implementa o algoritmo de busca binária, que possui complexidade de tempo $O(\log n)$, enquanto a função `funcao1` implementa uma busca linear, que possui complexidade de tempo $O(n)$. No pior caso, a busca binária é mais eficiente do que a busca linear, portanto a afirmação é falsa.

III. A função `funcao2` implementa uma estratégia iterativa na concepção do algoritmo.

Isso está incorreto. A função `funcao2` implementa o algoritmo de busca binária de forma recursiva, não iterativa. Portanto, a afirmação é falsa.

É correto o que se afirma em:

IV. I, apenas.

V. III, apenas.

VI. I e II, apenas.

VII. II e III, apenas.

VIII. I, II e III.

Justifique sua resposta.

24. Um usuário precisa implementar o controle de uma matriz de leds com 8×8 elementos. Para isso, ele criou um programa em C dotado de uma matriz da forma para armazenar os estados dos leds. Como existem apenas dois estados possíveis para os leds (aceso ou apagado), ele assumiu que leds acesos seriam denotados pelo inteiro "1" nessa matriz e leds apagados seriam denotados pelo inteiro "0".

```
unsigned char m[8][8];
```

Ocorre que a função que controla os leds exige que a informação que controla a matriz seja enviada via porta serial usando uma função que **recebe um único inteiro não sinalizado de 64 bits**, da forma `send(unsigned long estado)`. Nesse inteiro, os bytes mais significativos deverão guardar os estados das linhas iniciais da matriz de leds, enquanto os bytes menos significativos devem guardar os estados das linhas finais da matriz. Assim, é necessário que cada estado previsto na matriz `m` seja codificado em um bit correspondente na variável enviada pela função.

Crie um programa em linguagem C para realizar essa codificação e explique na forma de comentários como sua codificação da matriz `m` na variável de 64 bits foi realizada.

25. Um programador precisa desenvolver uma aplicação em linguagem C para manipular matrizes capazes de armazenar representações de modelos tridimensionais.

Entende-se que o tamanho da matriz é definido pelo usuário e esta deve ser alocada dinamicamente usando `malloc()` em tempo de execução. O processo de criar um modelo na matriz consiste em atribuir aos seus elementos os valores inteiros "1" ou "0" para simbolizar que há ou não parte do modelo naquela posição. Uma analogia para o modelo seria que a criação funciona como no jogo "Minecraft", onde "0" representaria a ausência de objeto e "1" representaria a presença de objetos.

Nesta aplicação, as matrizes devem ser definidas como tipos de dados `int`. Isto posto, pede-se que o programador prepare os algoritmos de alocação dinâmica (usando `malloc()/free()`) para guardar os dados da matriz tridimensional e crie um programa de testes para verificar se a sua implementação foi realizada corretamente de modo a garantir as seguintes condições:

- O usuário do programa de testes deverá poder fornecer o tamanho da matriz tridimensional que deseja manipular, inserindo as dimensões da altura, largura e profundidade desta.
- O usuário do programa de testes poderá solicitar a impressão de um dos planos da matriz tridimensional.
- O usuário do programa de testes poderá modificar o estado de um dos elementos da matriz. Insira no seu código comentários para indicar como as posições da matriz poderão ser acessadas.