Lista de Exercícios (Ponteiros)

Resolva os exercícios abaixo, obedecendo às seguintes instruções:

- Crie arquivos chamados `questao01.c`, `questao02.c`, ..., `questao25.c` com a resposta de cada questão
- Coloque-os em um diretório chamado `listaponteiros`.
- Compacte a pasta `listaponteiros` em um arquivo `listaponteiros.zip` e envie para o SIGAA.
- Questões enviadas sem resposta ou com resposta incompleta não serão corrigidas.
- Apenas serão corrigidas listas que seguirem esse padrão.
- Listas enviadas fora desse padrão terão nota ZERO atribuída.
- Será agendada uma data para a realização da prova oral, onde o aluno deverá explicar o código desenvolvido, a ser sorteado entre as questões enviadas pelo aluno.
- As questões de 1 a 25 valem `8/25` pontos cada.
- As questões de 26 a 30 valem `2/5` pontos cada.
- **CASO O ALUNO NÃO CONSIGA EXPLICAR A SUA PRÓPRIA QUESTÃO SORTEADA, SERÁ ATRIBUÍDA NOTA ZERO À LISTA. ASSEGURE-SE DE QUE SABE O QUE ESTÁ ENTREGANDO.**

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:

   - `p == &i;`
   - `*p - *q;`
   - `**&p;`
   - `3 - *p/(*q) + 7;`

2. Mostre o que será impresso pelo 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);
   ```

```
}
```

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

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

4. Determine o que será mostrado pelo seguinte program nos trechos `(a)` até `(n)`. Compile-o, execute-o e verifique se foram obtidas as respostas esperadas. Justifique o porque de cada uma.

```c
#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);
```

```c
  /* (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. Um programador construiu o seguinte código e esperava na saída o texto `gostinho`, mas não obteve essa saída. Justifique o porquê de não ter obtido o resultado esperado.

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

void funcao(char** str){
    str++;
}

int main(){
    char *str = (void *)malloc(50*sizeof(char));
    strcpy(str, "Agostinho");
    funcao(&str);
    puts(str);
    free(str);
    return 0;
}
```

6. Assumindo que um inteiro tem 4 bytes de tamanho, qual será a saída do seguinte programa? Justifique sua resposta.

```c
#include <stdio.h>
void funcao(char **p){
    char *t;
    t = (p += sizeof(int))[-1];
    printf("%s\n", t);
}
int main(){
    char *a[] = { "ab", "cd", "ef", "gh", "ij", "kl"};
    funcao(a);
    return 0;
}
```

7. Determine o que será mostrado pelo seguinte programa. Compile-o, execute-o e explique se foram obtidas as respostas esperadas. Justifique o porque de cada uma.

```c
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");
  }
}
```

8. Assumindo que `pulo[]` é um vetor do tipo `int`, qual das seguintes expressões referenciam o valor do terceiro elemento do vetor?
   - `*(pulo + 2);`
   - `*(pulo + 4);`
   - `pulo + 4;`
   - `pulo + 2;`
9. Considerando a declaração

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

   quais das seguintes expressões são válidas? Justifique.

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

10. O que fazem os seguintes programas em C?

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

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

11. Um programador pretendia armazenar em um struct (acessível através de um ponteiro) o nome de um usuário e um valor inteiro associado e preparou o seguinte programa. Há algum erro presente no código? Se sim, qual é ele e como pode ser corrigido?

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

12. Qual será a saída do seguinte programa

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

13. 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.

14. Justifique porque a saída do seguinte programa é mostrado o valor `19`

```c
#include <stdio.h>
int f(int a, int *pb, int **ppc) {
  int b, c;
  **ppc += 1;
  c = **ppc;
  *pb += 2;
  b = *pb;
  a += 3;
```

```
  return a + b + c;
}

void main() {
  int c, *b, **a;
  c = 5;
  b = &c;
  a = &b;
  printf("%d\n", f(c, b, a));
  getchar();
}
```

15. O que será mostrado na tela pelo seguinte programa? Justifique sua resposta.

```
#include <stdio.h>
int main(){
   unsigned int x[4][3] = {{1, 2, 3}, {4, 5, 6},
                           {7, 8, 9}, {10, 11, 12}};
   printf("%u, %u, %u", x+3, *(x+3), *(x+2)+3);
}
```

16. 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;
```

17. O que é memory leak? Crie um exemplo de programa em C que contenha um memory leak e proponha uma correção para o mesmo.
18. 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.
19. Implemente em linguagem C uma função em um programa de computador que leia n valores do tipo `float` do teclado e os apresente em ordem crescente. Utilize alocação dinâmica de memória para armazenar o conjunto de pontos lidos.

20. Reimplemente o programa da questão anterior utilizando a função `qsort()` do C. Comente o seu código, explicando o que faz cada uma das linhas.

21. 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.

22. 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 (com seu método de ordenação e com `qsort()`), considerando apenas arrays de elementos tipo `int` e compare os resultados obtidos. O que se conclui nesse caso?

23. 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, portanto processos de alocação dinâmica de memória são necessários. 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:

```
int n=4;
// ...
soma_vetores(vet1, vet2, resultado, n);
```

24. Crie uma função capaz de realizar multiplicação matricial da forma `C = AxB`. 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).

```
multiplica_matrizes(A, B, C, 2, 3, 4);
```

25. (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);
}
```

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

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

```
//(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;
}
```

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

26. (ENADE, 2023) Na programação de sistemas embarcados, algumas posições de memória servem para diferentes propósitos, não apenas para armazenar valores. Em algumas dessas memórias, cada um os 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.

```c
#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? Explique, PASSO-A-PASSO, os cálculos realizados pelo programa para chegar a esse resultado.

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

```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.

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

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

É correto o que se afirma em:

a. I, apenas.

b. III, apenas.

c. I e II, apenas.

d. II e III, apenas.

e. I, II e III.

Justifique sua resposta.

28. Considere o exemplo seguinte. Compile-o, execute-o e JUSTIFI, usando os recursos que julgar necessários, porque o programa imprime o resultado que imprime.

```
#include <stdio.h>

char *a[] = {"AGOSTINHO", "MEDEIROS", "BRITO", "JUNIOR"};
char **b[] = {a + 3, a + 2, a + 1, a};
char ***c = b;

int main() {
  printf("%s ", **++c);
  printf("%s ", *--*++c + 3);
  printf("%s ", *c[-2] + 3);
  printf("%s ", c[-1][-1] + 1);
  return 0;
}
```

29. Um usuário precisa implementar o controle de uma matriz de leds com `8 x 8` elementos. Para isso, ele criou um programa em C dotado de uma matriz da forma

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

para armazenar os estados dos leds. Como existem apenas dois estados possíveis para os leds (aceso ou apagado), ele assumiu que leds acessos (💡) seriam denotados pelo inteiro "1" nessa matriz e leds apagados (💡) seriam denotados pelo inteiro "0". Por exemplo, para a seguinte matriz de leds:

a matriz `m` seria codificada da seguinte forma:

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0

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.

Para a matriz acima, a variável de 64 bits enviada pela função `send()` deveria ser codificada da seguinte forma:

```
0101010110101010010101011010101001010101101010101010101010101010
```

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.

30. 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:

    a. 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.
    b. O usuário do programa de testes poderá solicitar a impressão de um dos planos da matriz tridimensional.
    c. 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.