Lista de Exercícios ponteiros e alocação de memória.

1.) Qual o valor de x, y e p no final da execução do programa abaixo?

```
...
int x, y, *p;
y = 0;
p = &y;
x = *p;
x = 4;
(*p)++;
x--;
(*p) += x;
...
```

Resultado: x = 3 y = 4 p = endereço de y.

2.) A função em C analise(&i, &j) tem um problema. Qual é? Antes da chamada do método, temos a seguinte linha de comando: int i=6, j=10

```
void misterio(int *p, int *q)
{
 int *temp;
 *temp = *p;
 *p = *q;
 *q = *temp;
}
```

Resultado: O endereço apontado por *temp nunca foi definido

3.) Qual o resultado obtido com a execução destas funções em C ?

a.)

```
void imprime_primeiro(int *vet)
{
 printf("Valor: %d\n", vet[0]);
}
int main(void)
{
 int vet[5] = {1, 2, 3, 4, 5};
 imprime_primeiro(vet);
 return 0;
}
```

Resultado: Valor = 1.

b.)

```
void imprime_primeiro(int *vet)
{
 printf("Valor: %d\n", vet[0]);
}
int main(void)
{
 int vet[5] = {1, 2, 3, 4, 5};
 imprime_primeiro(&vet[2]);
 return 0;
}
```

Resultado : valor = 3

c.)

```
int* metade (int *vet, int n)
{
 return &vet[(int)(n/2)];
}
int main(void)
{
 int vet[6] = {1, 2, 3, 4, 5, 6};
 int *v = metade (vet, 6);
 printf("Valor: %d\n", v[0]);
 return 0;
}
```

Resultado : valor = 4

d.)

```
int main(void)
{
 int vet[6] = {1, 2, 3, 4, 5, 6};
 printf("Valor1: %d\n", vet);
 printf("Valor2: %d\n", *vet);
 printf("Valor3: %d\n", *(vet + 2));
 return 0;
}
```

Resultado :

Valor1: endereço de memória de vet[0]
Valor2: 1
Valor3: 3

4.) Implemente e explique o código em C abaixo que aborda ponteiros apontando para strings

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

int strtam(char *s);

void main(void)
{
   char *lista="1234567890";

   printf("O tamanho do string \"%s\" e %d caracteres.\n", lista, strtam(lista));
   printf("Acabou.");
}

int strtam(char *s){

   int tam=0;

         while(*(s + tam++) != '\0');
   return tam-1;
}
```

5.) Implemente e explique o código em C abaixo:

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

int main(void)
{
      float vetor[] = { 1.0, 2.0, 3.0, 4.0, 5.0 };
      float *p1, *p2;

      p1 = &vetor[2];                  /* endereco do terceiro elemento */
      p2 = vetor;            /* endereco do primeiro elemento */

   printf("Diferenca entre ponteiros %d\n", p1-p2);
      return 0;
}
```

6.) Implemente e explique o código em C abaixo:

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

int main (void)
{
      char *c, *v, a, b;

      scanf("%c %c", &a, &b);
      c = &a;
      v = &b;
      if (c == v)
      {
          printf("As variáveis estao na mesma posicao.\n");       }
      else
      {
          printf("As variaveis nao estao na mesma posicao.\n"); }
      return 0;
}
```

7.) Implemente o programa abaixo, verifique e explique detalhadamente o que está sendo impresso.

```
#include < stdio .h >
int main (void) {
int var ; // var e um inteiro
int * varPtr ; // varPtr e um ponteiro para inteiros
var = 7;
varPtr = & var ; // varPtr aponta para onde esta var
printf ("O endereco de var e %p\n"
"O valor de varPtr e %p\n", &var , varPtr );
printf ("O valor de var e %d\n"
 "O valor que varPtr aponta e %d\n",var , * varPtr );
printf (" Mostrando que * e & sao complementares \n" " &* varPtr = %p\n" " *& varPtr =
%p\n", &* varPtr , *& varPtr );
return 0;
}
```

8.) Dado o problema abaixo, verifique a diferença na notação de ponteiros em p2 = &j e *p2 = temp. Verifique o que sera impresso!

```
#include < stdio .h >
#include < stdlib .h >
int main (void) {
int i = 10 , j = 20;
int temp ;
int * p1 , * p2 ;
p1 = &i; /* p1 recebe endereco de i */
p2 = &j /* p2 recebe endereco de j */
temp = * p1 ; /* conteudo apontado por p1 para temp */
* p1 = * p2 ; /* conteudo apontado por p2 para o apontado p1 */
* p2 = temp ; /* conteudo apontado por p1 para p2 */
printf ("%d %d\n", i , j);
return 0;
}
```

9.) Implemente e explique o código C abaixo:

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

#define N 10

void LeNumeros (int *lista);
void Ordena (int *lista);
void Imprime (int *lista);
int Insere (int *lista, int numero);

int main(void)
{
        int *lista;         /* lista de numeros */
        int numero;

        lista = (int *) malloc(sizeof(int)*N);

        if(!lista)
        {
                printf("Nao consegui alocar espaco.");
    return 1;;
  }
        LeNumeros(lista);
        Imprime(lista);
        Ordena(lista);
  Imprime(lista);
        printf("Forneca um numero. "); scanf("%d", &numero);
        if (Insere(lista, numero) == -1)
        {
                printf("Nao ha espaco.\n");
        }
  Imprime(lista);
        return 0;
}
void LeNumeros(int *lista)
{
        int i=0, numero;
        do
        {
                printf("Entre com o numero ");
                scanf("%d", &numero);
    *(lista+i) = numero;
    i++;
        } while(numero >= 0);
}

void Ordena(int *lista)
{
        int i, temp, trocou, elem=0;
```

```
        while(*(lista+elem)>= 0)
        {
        elem++;
        }
        printf("elem %d\n", elem);
        elem--;
        do
        {
                trocou = 0;
                for (i=0; i<elem; i++)
                {
                        if (*(lista+i) > *(lista+i+1))
                        {
                                trocou = 1;
                                temp = *(lista+i);
                                *(lista+i) = *(lista+i+1);
          *(lista+i+1) = temp;
        }
                }
   } while (trocou);
}

void Imprime (int *lista)
{
        int i=0;

        printf("imprime\n");
        while (*(lista + i) >= 0)
        {
                printf("Elemento %d = %d\n", i, *(lista+i));
                i++;
   }
}

int Insere (int *lista, int numero)
{
        int pos=0, i;
        i=N-1;

        printf("Insere\n");
        while(*(lista+pos) < numero && *(lista+pos) != -1)
        {
                pos++;  }
        printf("pos = %d\n", pos);
        if (pos == N-1)
        {
                return -1;    }
        while (i != (pos-1)) *(lista+i+1) = *(lista+i--);
        *(lista+pos) = numero;
        return 0;
}
```

10.) Este exercícios tem como objetivo exemplificar a permutabilidade de arrays e ponteiros. Para isto, no código abaixo são definidas duas funções de cópia de string — copy1 e copy2. As duas funções copiam uma string (possivelmente, um array de caracteres) para um array de caracteres. Elas realizam a mesma tarefa, porém, são implementadas de formas diferentes.

```c
/* Fig. 7.21: fig07_21.c
   Copiando uma string usando notação de array e notação de ponteiro. */
#include <stdio.h>

void copy1( char * const s1, const char * const s2 ); /* protótipo */
void copy2( char *s1, const char *s2 ); /* protótipo */

int main( void )
{
   char string1[ 10 ]; /* cria array string1 */
   char *string2 = "Olá"; /* cria um ponteiro para uma string */
   char string3[ 10 ]; /* cria array string3 */
   char string4[] = "Adeus"; /* cria um ponteiro para uma string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );
```

Figura 7.21 ■ Copiando uma string usando a notação de array e a notação de ponteiro. (Parte 1 de 2.)

```c

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */

/* copia s2 para s1 usando notação de array */
void copy1( char * const s1, const char * const s2 )
{
   int i; /* contador */

   /* loop pelas strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* não faz nada no corpo */
   } /* fim do for */
} /* fim da função copy1 */

/* copia s2 para s1 usando notação de ponteiro */
void copy2( char *s1, const char *s2 )
{
   /* loop pelas strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* não faz nada no corpo */
   } /* fim do for */
} /* fim da função copy2 */
```

```
string1 = Olá
string3 = Adeus
```

Figura 7.21 ■ Copiando uma string usando a notação de array e a notação de ponteiro. (Parte 2 de 2.)

11.)Faça um programa que leia dois valores inteiros e chame uma função que receba estes 2 valores (passagem de valores por referência) de entrada e retorne o maior valor na primeira variável e o menor valor na segunda variável. Escreva o conteúdo das 2 variáveis na tela.

12.) Elaborar um programa que leia dois valores inteiros (A e B). Em seguida faça uma função (passagem de parâmetros por referência) que retorne a soma do dobro dos dois números lidos. A função deverá armazenar o dobro de A na própria variável A e o dobro de B na própria variável B.

13.) Crie um programa que contenha uma função que permita passar por referência dois números inteiros A e B. A função deverá calcular a soma entre estes dois números e armazenar o resultado na variável A. Esta função não deverá possuir retorno, mas deverá modificar o valor do primeiro parametro. Imprima os valores de A e B na função principal.

14.) Crie um programa que contenha um array de float contendo 10 elementos. Imprima o endereço de cada posição desse array.

15.) Crie um programa que contenha uma matriz de float contendo 3 linhas e 3 colunas. Imprima o endereço de cada posição dessa matriz.

16.) Crie um programa que contenha um array de inteiros contendo 5 elementos. Utilizando apenas aritmética de ponteiros, leia esse array do teclado e imprima o dobro de cada valor lido.

17.) Crie um programa que contenha um array contendo 5 elementos inteiros. Leia esse array do teclado e imprima o endereço das posições contendo valores pares.

18.) Faça um programa que leia três valores inteiros e chame uma função que receba estes 3 valores de entrada e retorne eles ordenados, ou seja, o menor valor na primeira variável, o segundo menor valor na variável do meio, e o maior valor na ´ ultima variável. A função deve retornar o valor 1 se os três valores forem iguais e 0 se existirem valores diferentes. Exibir os valores ordenados na tela.

19.) Elabore uma função que receba duas strings como parâmetros e verifique se a segunda string ocorre dentro da primeira. Use aritmética de ponteiros para acessar os caracteres das strings.

20.) Crie uma função que receba dois parâmetros: um array e um valor do mesmo tipo do array. A função dever ´a preencher os elementos de array com esse valor. Não utilize índices para percorrer o array, apenas aritmética de ponteiros.

21.) Crie uma função que receba como parâmetro um array e o imprima. Não utilize índices para percorrer o array, apenas aritmética de ponteiros.

22.) Considere a seguinte declaração: int A, *B, **C, ***D; Escreva um programa que leia a variável a e calcule e exiba o dobro, o triplo e o quádruplo desse valor utilizando apenas os ponteiros B, C e D. O ponteiro B deve ser usada para calcular o dobro, C o triplo e D o quádruplo.

23). Escreva uma função que dado um número real passado como parâmetro, retorne a parte inteira e a parte fracionária deste número. Escreva um programa que chama esta função.
Protótipo: void frac(float num, int* inteiro, float* frac);

24.) Escreva uma funcão que aceita como parametro um array de inteiros com N valores, e determina o maior elemento do array e o numero de vezes que este elemento ocorreu no array. Por exemplo, para um array com os seguintes elementos: 5, 2, 15, 3, 7, 15, 8, 6, 15, a função deve retorna para o programa que a chamou o valor 15 e o número 3 (indicando que o numero 15 ocorreu 3 vezes). A função deve ser do tipo void.

25.) Implemente uma função que calcule a ´area da superfície e o volume de uma esfera de raio R. Essa função deve obedecer ao protótipo:
void calc_esfera(float R, float *area, float *volume)

A ´area da superfície e o volume são dados, respectivamente, por:
A = 4*p* R2
V = 4/3 * p * R3

26.)Implemente e analise este código em C que aborda ponteiros para ponteiros.

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

void main () {

   int **matriz;   /* ponteiro para os ponteiros de cada uma das linhas */
   int lin, col;   /* número de linhas e colunas */
   int i, j;
   int linha1, linha2; /* linhas da matriz que serao trocadas */
   char linha[80]; /* linha de caracteres com os dados */
   int *temp;

   do {
     puts("Qual o numero de linhas?");
     gets(linha);
     lin = atoi(linha);
   } while (lin<=0);
   printf("Numero de linhas = %d\n", lin);
   printf("Alocando espaço para armazenar os ponteiros para as linhas.\n");
   matriz = (int **) malloc (lin * sizeof(int *));
   if (!matriz) {
     puts("Nao há espaço para alocar memória");
     exit(1);
   }

   do {
     puts("Qual o numero de colunas?");
     gets(linha);
     col = atoi(linha);
   } while (col<=0);
```

```
printf("Numero de colunas = %d\n", lin);
printf("Alocando espaço para armazenar os vetores de linhas.\n");
for (i=0; i<lin; i++)
{
  *(matriz +i) = (int *) malloc(col * sizeof (int));
  if (! *(matriz+i) ){
    printf("Não há espaço para alocar a linha %d", i);
    exit(1);
  }
}
puts("Entre com os dados");
for (i=0; i<lin; i++)
{
  printf("Entre com a linha %d\n", i);
  for (j=0; j<col; j++)
  {
     printf("Elemento %d %d\n", i, j);
     scanf("%d", *(matriz +i) +j);
  }
}
puts("Dados lidos");
for (i=0; i<lin; i++)
{
  for (j=0; j<col; j++)
  {
     printf("%7d ", *(*(matriz +i) +j));
  }
  printf("\n");
}
getchar();

do {
  puts("Qual a primeira linha a ser trocada?");
  gets(linha);
  linha1=atoi(linha);
} while (linha1 >=lin);

do {
  puts("Qual a segunda linha a ser trocada?");
  gets(linha);
  linha2=atoi(linha);
} while (linha2 >=lin);

temp = *(matriz + linha1);
*(matriz + linha1) = *(matriz + linha2);
*(matriz + linha2) = temp;
puts("Dados trocados.");
for (i=0; i<lin; i++)
{
  for (j=0; j<col; j++)
  {
     printf("%7d ", *(*(matriz +i) +j));
```

```
        }
        printf("\n");
    }
}
```

26) Escreve um programa em C que simule o Jogo da velha. Você ira resolver este exercício de duas formas. Na primeira forma você resolverá o problema sem manipular ponteiros, usando apenas a matriz 3x3 na passagem de parâmetros para as funções. Na segunda forma você utilizará ponteiros para manipular a matriz e também criará a matriz dinamicamente. Segue abaixo o conjunto "sugestivo" de funções que você utilizará em seu programa. Voce também poderá criar outras funções.

```
void zeraTabuleiro(int tabuleiro[][DIM]);
void exibeTabuleiro(int tabuleiro[][DIM]);
void jogar(int tabuleiro[][DIM]);
int checaLocal(int tabuleiro[][DIM], int linha, int coluna);
int checaLinha(int tabuleiro[][DIM]);
int checaColuna(int tabuleiro[][DIM]);
int checaDiagonal(int tabuleiro[][DIM]);
int checaEmpate(int tabuleiro[][DIM]);
int checaTermino(int tabuleiro[][DIM], int vez);
void jogada(int tabuleiro[][DIM]);
```

resolução da primeira forma:

```
#include <stdio.h>
#define DIM 3
int vez;

int menu(void);
void clear(void);
void zeraTabuleiro(int tabuleiro[][DIM]);
void exibeTabuleiro(int tabuleiro[][DIM]);
void jogar(int tabuleiro[][DIM]);
int checaLocal(int tabuleiro[][DIM], int linha, int coluna);
int checaLinha(int tabuleiro[][DIM]);
int checaColuna(int tabuleiro[][DIM]);
int checaDiagonal(int tabuleiro[][DIM]);
int checaEmpate(int tabuleiro[][DIM]);
int checaTermino(int tabuleiro[][DIM], int vez);
void jogada(int tabuleiro[][DIM]);

int main(void)
{
  int tabuleiro[DIM][DIM],
     continuar;

  do
  {
     vez=1;
     continuar = menu();
     if(continuar == 1)
        jogar(tabuleiro);

  }while(continuar);

  return 0;
```

```
}

int menu(void)
{
   int opcao;

      printf("\t\tJogo da Velha \n");
      printf("\n1.Jogar\n");
      printf("0.Sair\n");
      printf("\nOpcao: ");

      scanf("%d", &opcao);

      switch(opcao)
      {
        case 1:
        case 0:
            break;
        default:
            clear();
            printf("Opcao invalida. Tente de novo!\n");
      }

   return opcao;
}

void clear(void)
{
   int count=0;

   while(count != 100)
   {
      putchar('\n');
      count++;
   }
}

void zeraTabuleiro(int tabuleiro[][DIM])
{
   int linha, coluna;
   for(linha = 0 ; linha < DIM ; linha++)
      for(coluna = 0 ; coluna < DIM ; coluna++)
         tabuleiro[linha][coluna] = 0;
}

void exibeTabuleiro(int tabuleiro[][DIM])
{
   int linha, coluna;
   putchar('\n');

   for(linha = 0 ; linha < DIM ; linha++)
   {
```

```
      for(coluna = 0 ; coluna < DIM ; coluna++)
      {
         if(tabuleiro[linha][coluna] == 0)
            printf("    ");
         else
            if(tabuleiro[linha][coluna] == 1)
               printf("  X ");
            else
               printf("  O ");

         if(coluna != (DIM-1))
            printf("|");
      }
      putchar('\n');
   }
   putchar('\n');
}

void jogar(int tabuleiro[][DIM])
{
   int continua;
   zeraTabuleiro(tabuleiro);

   do
   {
      clear();
      exibeTabuleiro(tabuleiro);
      jogada(tabuleiro);

   }while(checaTermino(tabuleiro, vez) != 1);
}


int checaLocal(int tabuleiro[][DIM], int linha, int coluna)
{
   if(linha < 0 || linha > (DIM-1) || coluna < 0 || coluna > (DIM-1) || tabuleiro[linha][coluna]
!= 0)
      return 0;
   else
      return 1;
}

int checaLinha(int tabuleiro[][DIM])
{
   int linha, coluna,
      soma;

   for(linha = 0 ; linha < DIM ; linha++)
   {
      soma=0;

      for(coluna = 0 ; coluna < DIM ; coluna++)
```

```
        soma += tabuleiro[linha][coluna];

      if(soma==DIM || soma == (-1)*DIM)
        return 1;
   }

   return 0;
}

int checaColuna(int tabuleiro[][DIM])
{
   int linha, coluna,
      soma;


   for(coluna = 0 ; coluna < DIM ; coluna++)
   {
      soma=0;

      for(linha = 0 ; linha < DIM ; linha++)
        soma += tabuleiro[linha][coluna];

      if(soma==DIM || soma == (-1)*DIM)
        return 1;
   }

   return 0;
}

int checaDiagonal(int tabuleiro[][DIM])
{
   int linha,
      diagonal_principal=0,
      diagonal_secundaria=0;

   for(linha = 0 ; linha < DIM ; linha++)
   {
      diagonal_principal += tabuleiro[linha][linha];
      diagonal_secundaria += tabuleiro[linha][DIM-linha-1];
   }

   if(diagonal_principal==DIM || diagonal_principal==(-1)*DIM ||
     diagonal_secundaria==DIM || diagonal_secundaria==(-1)*DIM)
     return 1;

   return 0;
}

int checaEmpate(int tabuleiro[][DIM])
{
   int linha, coluna;
```

```
   for(linha = 0 ; linha < DIM ; linha++)
      for(coluna = 0 ; coluna < DIM ; coluna++)
         if(tabuleiro[linha][coluna] == 0)
            return 0;

   return 1;
}

int checaTermino(int tabuleiro[][DIM], int vez)
{
   if(checaLinha(tabuleiro))
   {
      printf("Jogo encerrado. Jogador %d venceu !\n\n", (vez%2)+1);
      exibeTabuleiro(tabuleiro);
      return 1;
   }

   if(checaColuna(tabuleiro))
   {
      printf("Jogo encerrado. Jogador %d venceu !\n\n", (vez%2)+1);
      exibeTabuleiro(tabuleiro);
      return 1;
   }

   if(checaDiagonal(tabuleiro))
   {
      printf("Jogo encerrado. Jogador %d venceu !\n\n", (vez%2)+1);
      exibeTabuleiro(tabuleiro);
      return 1;
   }

   if(checaEmpate(tabuleiro))
   {
      printf("Jogo encerrado. Ocorreu um empate! !\n\n");
      exibeTabuleiro(tabuleiro);
      return 1;
   }

   return 0;
}

void jogada(int tabuleiro[][DIM])
{
   int linha, coluna;
   vez++;
   printf("\n--> Jogador %d \n", (vez % 2) + 1);

   do
   {
      printf("Linha: ");
      scanf("%d", &linha);
      linha--;
```

```
      printf("Coluna: ");
      scanf("%d", &coluna);
      coluna--;

      if(checaLocal(tabuleiro, linha, coluna) == 0)
        printf("Posicao ocupada ou inexistente, escolha outra.\n");

   } while(checaLocal(tabuleiro, linha, coluna) == 0);

   if(vez%2)
     tabuleiro[linha][coluna] = -1;
   else
     tabuleiro[linha][coluna] = 1;
}
```

27)Escreva um programa que possua três variáveis (dos tipos char, int e float). Solicite ao usuário para digitar valores para estas três variáveis e imprima os endereços de memória onde estas variáveis estão armazenadas. Execute várias vezes este programa. O resultado exibido na tela é igual em todas as execuções? Por que?

28.) Insira três ponteiros ao código elaborado no exercício 1 (um ponteiro para char, um ponteiro para int e um ponteiro para float). Associe cada uma das variáveis aos respectivos ponteiros. Altere seu código para receber os valores digitados pelo usuário através dos ponteiros. Após receber os valores, imprima o conteúdo das variáveis na tela.

29.) Faça um programa que possua três variáveis do tipo char (a, b, c) e um ponteiro para char (pc). Solicite ao usuário para informar os valores para as três variáveis do tipo caracter. Imprima o conteúdo de cada uma das variáveis utilizando apenas o ponteiro pc.

30.) Faça um programa que receba dois números inteiros (a, b) e apresente a soma destes valores (res). Associe os inteiros (a,b, res) aos ponteiros (pa, pb, pres). Faça a entrada dos dois números e o cálculo da soma (que deve ser armazenado em res) usando apenas os ponteiros (pa, pb, pres). Apresente o resultado da soma armazenado em res.

31.)Faça um programa que possua um vetor (vi) com 10 números inteiros. Gere valores aleatórios para preencher o vetor vi. Na sequência, defina um vetor de ponteiros para números inteiros (vponteiro), também com 10 posições, e armazene o endereço de cada um dos elementos de vi neste vetor de ponteiros vponteiro. Faça um laço para apresentar os endereços de memória de cada um dos 10 valores e, também, o conteúdo de cada posição dos vetores.

32.) Criar um vetor de inteiros e incluir valores aleatórios em suas posições. Usando aritmética de ponteiros para o percurso no vetor, faça:
(a) Encontrar o menor elemento do vetor.
(b) Calcular a média entre os elementos.
(c) Encontrar o número de elementos negativos.
(d) Imprimir os elementos em ordem invertida.
(e) Usar a função printf para retornar o número de bytes alocados a cada posição no vetor. Como as posições no vetor são contínuas, a diferença entre dois endereços consecutivos permite descobrir o tamanho de cada posição.

33.) Construa um vetor de números reais e percorra todo o vetor usando dois ponteiros : um começando do início do vetor e outro do final.

34.) Analise a estrutura de ponteiros para ponteiros implementando e executando o programa abaixo.

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

int main (void) {
int ** matriz ; /* ponteiro para os ponteiros */
int lin , col ; /* número de linhas e colunas */
int i , j;
int linha1 , linha2 ; /* linhas que serao trocadas */
char linha [80]; /* linha de caracteres com os dados */
int * temp ;
puts (" Qual o numero de linhas ?");
gets ( linha ); lin = atoi ( linha );
matriz = ( int **) malloc( lin * s i zeof ( int *) );
i f (! matriz ) {
puts (" Nao há espaço para alocar memória ");
return 1;
}
puts (" Qual o numero de colunas ?");
gets ( linha );
ol = atoi ( linha );
for (i =0; i< lin ; i ++) {
*( matriz +i) = ( int *) malloc (col * s i zeof ( int ));
i f (! *( matriz +i) ) {
printf (" Sem espaço para alocar a linha %d", i);
return 1; }
}
puts (" Entre com os dados");}
for (i =0; i< lin ; i ++) {
        printf (" Entre com a linha %d \n", i);
        for (j =0; j<col ; j ++) {
                printf (" Elemento %d %d\n" , i , j);
                scanf ("%d", *( matriz +i) +j); }
}
puts (" Qual a primeira linha a ser trocada ?");
gets ( linha ); linha1 = atoi ( linha );
puts (" Qual a segunda linha a ser trocada ?");
gets ( linha ); linha2 = atoi ( linha );
temp = *( matriz + linha1 );
*( matriz + linha1 ) = *( matriz + linha2 );
*( matriz + linha2 ) = temp ;
puts (" Dados trocados .");
for (i =0; i< lin ; i ++) {
        for (j =0; j< col ; j ++) {
                printf ("%7 d ", *(*( matriz +i) +j)); }
printf ("\n");
}
return 0;
}
```

35.) Analise a estrutura de ponteiros para ponteiros usando funções implementando e executando o programa abaixo.

```
#include < stdio .h>
#include < stdlib .h >}
int ** aloca_linhas ( int );
void aloca_colunas ( int **, int , int );
void le_dados ( int **, int , int );
void imprime_matriz ( int **, int , int );
void troca_linhas ( int **, int , int );
int main (void) {
int ** matriz ;
int lin , col ;& nbsp ;& nbsp ;
int linha1 , linha2 ;
char linha [80];
puts (" Qual o numero de linhas ?");
gets ( linha ); lin = atoi ( linha );
matriz = aloca_linhas ( lin );
puts (" Qual o numero de  colunas ?");
gets ( linha );
col = atoi ( linha );
printf (" Alocando espaço para linhas .\ n");
aloca_colunas ( matriz , lin ,col );
le_dados ( matriz , lin , col );
imprime_matriz ( matriz , lin , col );
puts (" Qual a primeira linha a ser trocada ?");
gets ( linha ); linha1 = atoi ( linha );
puts (" Qual a segunda linha a ser trocada ?");
gets ( linha ); linha2 = atoi ( linha );
troca_linhas ( matriz , linha1 , linha2 );
imprime_matriz ( matriz , lin ,col );
return 0;
}
int ** aloca_linhas ( int lin ) {
int ** m;
m = ( int **) malloc( lin * s i zeof ( int *) );
i f (! m) {
puts (" Sem espaço para alocar memória ");
return 1;
}
return m;
}
void aloca_colunas ( int ** matriz , int lin , int col ) {
int i;
for (i =0; i< lin ; i ++) {
*( matriz +i) = ( int *) malloc(col * s i zeof ( int ));
i f (! *( matriz +i) ) {
printf (" Sem espaço para linha %d", i);
return 1;
}
}
}
```

```
void le_dados ( int ** matriz , int lin , int col ) {
int i , j;
puts (" Entre com os dados");
for (i =0; i< lin ; i ++) {
printf (" Entre com a linha %d\n", i);
for (j =0; j< col ; j ++) {
printf (" Elemento %d %d\n" , i , j);
scanf ("%d", *( matriz +i) +j);
}
}
}
void imprime_matriz ( int ** matriz , int lin , int col ) {
int i , j;
for (i =0; i< lin ; i ++) {
for (j =0; j< col ; j ++) {
printf ("%7 d ", *(*( matriz +i) +j));
}
printf ("\n");
}
}
void troca_linhas ( int ** matriz , int linha1 , int linha2 ) {
int * temp ;
temp = *( matriz + linha1 );
*( matriz + linha1 ) = *( matriz + linha2 );
*( matriz + linha2 ) = temp ;
}
```

Alocação dinâmica de memória

Esta lista de exercícios contem na sua primeira parte exercícios resolvidos. Na segunda parte estão os exercícios propostos. Sugere se que os alunos implementem primeiro os exercícios resolvidos e depois resolvam os propostos.

36.) Implemente e analise este programa em C

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

int main(void)
{
        int *p1, *p2;
  float *f1;

  p1 = (int *) calloc(1, sizeof (int));
  p2 = (int *) calloc(1, sizeof (int));
  *p1 = 10;
  *p2 = 20;
  f1 = (float *) malloc (sizeof(float));
  *f1 = 3.14;
  printf("%d %d %f\n", *p1, *p2, *f1);
        return 0;
}
```

37.) Implemente e analise este programa em C

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#define LIN 3
#define COL 4

void main(void)
{
int *matriz;
int i, j;
        matriz = malloc(LIN*COL*sizeof(int));
        if (!matriz){
                printf("Nao consegui alocar a memoria suficiente.\n");
                exit(1);
        }
        for(i=0; i<LIN; i++){
                for (j=0; j<COL; j++){
                        printf("Elemento %d %d = ", i, j);
                        scanf("%d", matriz+(i*COL+j)*sizeof(int));
    }
        }
        for(i=0; i<LIN; i++){
                for (j=0; j<COL; j++){
                        printf("El%2d,%2d = %2d ", i, j, *(matriz+(i*COL+j)*sizeof(int)));
                }
                printf("\n");
        }
        printf("Qual elemento que imprimir? "); scanf ("%d %d", &i, &j);
        printf("elemento = %d\n", *(matriz+(i*COL+j)*sizeof(int)));
        free(matriz);
        printf("Acabou.");
}
```

38.) Implemente e analise este programa em C

```
#include <stdio .h>
#include <stdlib .h>
void LeVetor ( f loat *v, int tam );
f loat ProcuraMaior ( f loat *v , int tam , int * vezes );
int main (void) {
f loat *v , maior ;
int i , tam , vezes;
printf (" Qual o tamanho do vetor? ");
scanf ("%d", & tam );
v = ( f loat *) malloc ( tam * s i zeof ( f loat ));
i f (v) {
LeVetor (v, tam );
maior = ProcuraMaior (v, tam , & vezes );
printf (" Maior = %f e aparece %d vezes .\ n.", maior , vezes );
free (v);
}
else {
printf (" Não consegui alocar memoria .");
return 1;
}
return 0;
}
void LeVetor ( f loat *v, int tam ) {
int i;
for (i =0; i< tam ; i ++) {
printf (" Elemento %d ?", i);
scanf ("%f", v+i);
printf ("Li valor %f \n", *( v+i));
}
}
f loat ProcuraMaior ( f loat *v , int tam , int * vezes ) {
int i;
float maior ;
maior = v [0]; * vezes = 1;
for (i =1; i< tam ; i ++) {
i f (v[i] &gt ; maior ) {
maior = v[i ];
* vezes = 1;
}
else i f ( maior == v[i]) * vezes =* vezes +1;
}
return maior ;
}
```

39.) Este programa em C manipula ponteiros para strings. Implemente e analise o código

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

int strcop(char *d, char *o);

void main(void)

{
        char destino[20];
        char  *origem="string de origem";

        strcop(destino, origem);  /* copia o string origem para o destino */
        printf("%s\n", origem);
        printf("%s\n", destino);
           printf("Acabou.\n");
}

int strcop(char *d, char *o){
        while ((*d++ = *o++) != '\0');
        return 1;
}
```

40.) Implemente e analise o código em C.

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

#define LINHAS 5
#define COLUNAS 6

int main(void)
{
	int *linha[LINHAS];
	int i, j;

	printf("Comecei.\n");
	/* alocaespaco(linha); */
	for (i=0; i<LINHAS; i++){
		if (!(linha[i] =  (int *) malloc(COLUNAS*sizeof(int)))){
			printf("Nao consegui alocar o vetor %d.\n", i);
			exit(i);
		}
	}
	/* lelinhas(linha); */
	for (i=0; i<LINHAS; i++){
		printf("Entre com a linha %d.\n", i);
		for (j=0; j<COLUNAS; j++)
		{
			printf(" %d", j);
			scanf(" %d", &linha[i][j]);
		}
	}

	/* imprimelinhas(linha); */
	for (i=0; i<LINHAS; i++){
		printf("Linha %d \n", i);
		for (j=0; j<COLUNAS; j++)
		{
			printf("%d ", *(linha[i]+j));
		}
		printf("\n");
  }

	return 0;


}
```

41.) Implemente e analise este programa em C que aborda ponteiros para ponteiros utilizando funções

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


int **aloca_linhas(int );
void aloca_colunas (int **, int, int);
void le_dados(int **, int, int );
void imprime_matriz(int **, int, int );
void troca_linhas (int **, int, int);

void main () {

   int **matriz;   /* ponteiro para os ponteiros de cada uma das linhas */
   int lin, col;   /* número de linhas e colunas */
   //     int i, j;
   int linha1, linha2; /* linhas da matriz que serao trocadas */
   char linha[80]; /* linha de caracteres com os dados */
   // int temp;


   do {
     puts("Qual o numero de linhas?");
     gets(linha);
     lin = atoi(linha);
   } while (lin<=0);
   printf("Numero de linhas = %d\n", lin);

   printf("Alocando espaço para armazenar os ponteiros para as linhas.\n");
   matriz = aloca_linhas(lin);

   do {
     puts("Qual o numero de colunas?");
     gets(linha);
     col = atoi(linha);
   } while (col<=0);
   printf("Numero de colunas = %d\n", lin);

   printf("Alocando espaço para armazenar os vetores de linhas.\n");
   aloca_colunas(matriz, lin, col);


   le_dados(matriz, lin, col);

   puts("Imprimindo dados lidos");
   imprime_matriz(matriz, lin, col);

   getchar();

   do {
```

```
      puts("Qual a primeira linha a ser trocada?");
      gets(linha);
      linha1=atoi(linha);
    } while (linha1 >=lin);

    do {
      puts("Qual a segunda linha a ser trocada?");
      gets(linha);
      linha2=atoi(linha);
    } while (linha2 >=lin);

    troca_linhas(matriz, linha1, linha2);

    puts("Imprimindo dados trocados.");
    imprime_matriz(matriz, lin, col);

}

int **aloca_linhas(int lin) {

 int **m;
    m = (int **) malloc (lin * sizeof(int *));
    if (!m) {
      puts("Nao há espaço para alocar memória");
      exit(1);
    }
    return m;


}

void aloca_colunas(int **matriz, int lin, int col)
{
 int i;
    for (i=0; i<lin; i++)
    {
      *(matriz +i) = (int *) malloc(col * sizeof (int));
      if (! *(matriz+i) ){
        printf("Não há espaço para alocar a linha %d", i);
        exit(1);
      }
    }
}

void le_dados (int **matriz, int lin, int col)
{
 int i, j;
    puts("Entre com os dados");
    for (i=0; i<lin; i++)
    {
      printf("Entre com a linha %d\n", i);
      for (j=0; j<col; j++)
```

```
      {
         printf("Elemento %d %d\n", i, j);
         scanf("%d", *(matriz +i) +j);
      }
   }
}

void imprime_matriz (int **matriz, int lin, int col)
{
   int i, j;

   for (i=0; i<lin; i++)
   {
     for (j=0; j<col; j++)
     {
        printf("%7d ", *(*(matriz +i) +j));
     }
     printf("\n");
   }
}

void troca_linhas ( int **matriz, int linha1, int linha2)
{

 int *temp;

 temp = *(matriz + linha1);
 *(matriz + linha1) = *(matriz + linha2);
 *(matriz + linha2) = temp;
}
```

42.) Crie um programa que:
(a) Aloque dinamicamente um array de 5 números inteiros,
(b) Peça para o usuário digitar os 5 números no espaço alocado,
(c) Mostre na tela os 5 números,
(d) Libere a memória alocada.

43.) Faça um programa que leia do usuário o tamanho de um vetor a ser lido e faça a alocação dinâmica de memória. Em seguida, leia do usuário seus valores e mostre quantos dos números são pares e quantos são ímpares.

44.) Faça um programa que leia um número N e:
. Crie dinamicamente e leia um vetor de inteiro de N posições;
. Leia um número inteiro X e conte e mostre os múltiplos desse número que existem no vetor.

45.) Faça um programa que simule a memória de um computador: o usuário irá especificar o tamanho da memória, ou seja, quantos bytes serão alocados do tipo inteiro. Para tanto, a memória solicitada deve ser um valor múltiplo do tamanho do tipo inteiro. Em seguida, o usuário terá 2 opções: inserir um valor em uma determinada posição ou consultar o valor contido em uma determinada posição. A memória deve iniciar com todos os dados zerados.

46.) Escreva um programa que leia primeiro os 6 números gerados pela loteria e depois os 6 números do seu bilhete. O programa então compara quantos números o jogador acertou. Em seguida, ele aloca espaço para um vetor de tamanho igual a quantidade de números corretos e guarda os números corretos nesse vetor. Finalmente, o programa exibe os números sorteados e os seus números corretos.

47.) Faça um programa para armazenar em memória um vetor de dados contendo 1500 valores do tipo int, usando a função de alocação dinâmica de memória CALLOC:
(a) Faça um loop e verifique se o vetor contem realmente os 1500 valores inicializados com zero (conte os 1500 zeros do vetor).
(b) Atribua para cada elemento do vetor o valor do seu índice junto a este vetor.
(c) Exibir na tela os 10 primeiros e os 10 últimos elementos do vetor.

48.) Faça um programa que pergunte ao usu´ário quantos valores ele deseja armazenar em um vetor de double, depois use a função MALLOC para reservar (alocar) o espaço de memória de acordo com o especificado pelo usuário. Esse vetor deve ter um tamanho maior ou igual a 10 elementos. Use este vetor dinâmico como um vetor comum, atribuindo aos 10 primeiros elementos do vetor valores aleatórios (usando a funçãao rand) entre 0 e 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor.

49.) Construa um programa que leia o número de linhas e de colunas de uma matriz de números reais, aloque espaço dinamicamente para esta e a inicialize com valores fornecidos pelo usuário. Localize na matriz os três maiores números de uma matriz e mostre a linha e a coluna onde estão.

50.) Faça um programa que leia números do teclado e os armazene em um vetor alocado dinamicamente. O usuário irá digitar uma sequência de números, sem limite de quantidade. Os números serão digitados um a um e, sendo que caso ele deseje encerrar a entrada de dados, ele ira digitar o número ZERO. Os dados devem ser armazenados na memória deste modo:
. Inicie com um vetor de tamanho 10 alocado dinamicamente;

. Após, caso o vetor alocado esteja cheio, aloque um novo vetor do tamanho do vetor anterior adicionado espaço para mais 10 valores (tamanho N+10, onde N inicia com 10);
. Copie os valores j á digitados da área inicial para esta área maior e libere a memória da área inicial;
. Repita este procedimento de expandir dinamicamente com mais 10 valores o vetor alocado cada vez que o mesmo estiver cheio. Assim o vetor irá ser 'expandido' de 10 em 10 valores.

51.) Faça um programa que:
(a) Peça para o usuário entrar com o nome e a posição (coordenadas X e Y) de N cidades e as armazene em um vetor de estruturas (N ´e informado pelo usuário);
(b) Crie uma matriz de distâncias entre cidades de tamanho N x N;
(c) Calcule as distância entre cada duas cidades e armazene na matriz;
(d) Exiba na tela a matriz de distâncias obtida;
(e) Quando o usuário digitar o número de duas cidades o programa deverá retornar a distância entre elas.

52.) Implemente uma função em C que recebe dois vetores de inteiros v1 e v2, mais um inteiro N com o tamanho dos vetores. Sua função deve alocar e retornar um vetor de inteiros de tamanho N onde o elemento na posição i de v3 é a soma dos elementos na posição i de v1 e v2. Sua função deve ter a seguinte declaração:

int * soma_vetores(int *v1, int *v2, int N);

Para testar seu programa, crie uma função main() que chama sua função soma_vetores e imprime os valores somados

53.) Neste problema você deverá ler 15 valores colocá-los em 2 arrays conforme estes valores forem pares ou ímpares. Só que o tamanho de cada um dos dois arrays é de 5 posições. Então, cada vez que um dos dois arrays encher, você deverá imprimir todo o array e utilizá-lo novamente para os próximos números que forem lidos. Terminada a leitura, deve-se imprimir o conteúdo que restou em cada um dos dois arrays, imprimindo primeiro os valores do array ímpar. Cada array pode ser preenchido tantas vezes quantas for necessário. Utilize alocação dinâmica para criar os arrays.

54.) Faca um programa que multiplique duas matrizes. O programa deverá estar estruturado de maneira que:

a.) o usuário forneça as dimensões das matrizes (teste se as dimensões são compatíveis, isto e', se as matrizes podem ser multiplicadas);
b.) as matrizes sejam alocadas dinamicamente. Utilize a forma de ponteiro para ponteiros ;
c.) as matrizes sejam lidas pelo teclado (faca uma função para leitura das matrizes);
d.) as matrizes sejam, então, multiplicadas (faça uma função para a multiplicação);
e.) a matriz resultante seja apresentada em tela (faca uma função para apresentar a matriz na tela).

OBS:
a) Faca, tambem, alocacao dinamica da matriz resultante.
Suponha as matrizes A(mXn)

55.) Voce está jogando um jogo de tabuleiro. O jogo é bem simples, tem o objetivo de passar por todas as casas. É jogado com um dado normal de seis lados(1, 2, 3, 4, 5, 6). O valor obtido no dado será a quantidade de casas que será andada. Voce está no meio de uma partida, ou seja, já jogou uma certa quantidade de jogadas e se encontra a uma certa distância do final do jogo. Ele quer saber de quantas maneiras diferentes ele conseguirá chegar ou ultrapassar a casa que delimita o final do jogo.

Por exemplo, voce quer ganhar com duas jogadas e sua distância para ganhar da casa final é exatamente 11. Nesse cenário, as possíveis soluções são:

(6,6). Jogar o dado 2 vezes e obter o valor 6 ambas as vezes.
(6,5). Na primeira jogada obteve 6 e na sgunda 5.
(5,6). Na primeira obteve 5 e na segunda 6.
Logo, a resposta é 3. Perceba que mesmo (6,6) ultrapasse o valor desejado, se trata de um possível caminho.