Lista de Exercício – Estrutura de Dados 01

VETOR E MATRIZ

1 – Faça um programa que leia e armazene 5 valores inteiros em cada um dos vetores v1 e v2. A partir destes valores lidos, mostre na tela:

a) A soma dos elementos de cada vetor, nas respectivas posições.
b) A diferença dos elementos de cada vetor, nas respectivas posições.
c) O produto dos elementos de cada vetor, nas respectivas posições.
d) A divisão entre os elementos de cada vetor, nas respectivas posições.

```c
#include <stdio.h>

int ex1() {
  int v1[5], v2[5];
  int i;

  printf("Digite os valores do vetor v1:\n");
  for (i = 0; i < 5; i++) {
    scanf("%d", &v1[i]);
  }

  printf("Digite os valores do vetor v2:\n");
  for (i = 0; i < 5; i++) {
    scanf("%d", &v2[i]);
  }

  printf("Soma dos elementos:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", v1[i] + v2[i]);
  }
  printf("\n");

  printf("Diferença dos elementos:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", v1[i] - v2[i]);
  }
  printf("\n");

  printf("Produto dos elementos:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", v1[i] * v2[i]);
  }
  printf("\n");

  printf("Divisão dos elementos:\n");
  for (i = 0; i < 5; i++) {
    if (v2[i] != 0) {
      printf("%d ", v1[i] / v2[i]);
```

```
    } else {
      printf("Divisão por zero\n");
    }
  }
  printf("\n");

  return 0;
}
```

2 – Escreva um programa que lê uma matriz M[5][5] e mostre na tela:
a) A soma da linha 4 de M
b) A soma da coluna 2 de M
c) A soma da diagonal principal
d) A soma da diagonal secundária
e) A soma de todos os elementos da matriz

```c
#include <stdio.h>

int ex2() {
  int M[5][5];
  int i, j;
  int soma_linha_4 = 0, soma_coluna_2 = 0, soma_diagonal_principal = 0,
      soma_diagonal_secundaria = 0, soma_total = 0;

  printf("Digite os elementos da matriz M[5][5]:\n");
  for (i = 0; i < 5; i++) {
    for (j = 0; j < 5; j++) {
      scanf("%d", &M[i][j]);
    }
  }

  for (j = 0; j < 5; j++) {
    soma_linha_4 += M[3][j];
  }

  for (i = 0; i < 5; i++) {
    soma_coluna_2 += M[i][1];
  }

  for (i = 0; i < 5; i++) {
    soma_diagonal_principal += M[i][i];
  }

  for (i = 0; i < 5; i++) {
    soma_diagonal_secundaria += M[i][4 - i];
  }
```

```
  for (i = 0; i < 5; i++) {
    for (j = 0; j < 5; j++) {
      soma_total += M[i][j];
    }
  }

  printf("Soma da linha 4: %d\n", soma_linha_4);
  printf("Soma da coluna 2: %d\n", soma_coluna_2);
  printf("Soma da diagonal principal: %d\n", soma_diagonal_principal);
  printf("Soma da diagonal secundária: %d\n", soma_diagonal_secundaria);
  printf("Soma total: %d\n", soma_total);

  return 0;
}
```

ALGORITMOS DE ORDENAÇÃO

3 – Os algoritmos de ordenação servem para organizar uma lista de números ou palavras de acordo com a sua necessidade ou critério definido. Com base nisso, implemente utilizando a linguagem C, os algoritmos Bubble Sort, Insertion Sort e Selection Sort. Para isso, utilize o seguinte conjunto de dados: 1,6,8,2,4,9 e 5.

a) A soma dos elementos de cada vetor, nas respectivas posições.
b) A diferença dos elementos de cada vetor, nas respectivas posições.
c) O produto dos elementos de cada vetor, nas respectivas posições.
d) A divisão entre os elementos de cada vetor, nas respectivas posições.

```c
#include <stdio.h>

void bubbleSort(int* v, int n) {
   int fim, i, aux;
   for (fim = n - 1; fim > 0; fim--) {
      for (i = 0; i < fim; i++) {
         if (v[i] > v[i + 1]) {
            aux = v[i];
            v[i] = v[i + 1];
            v[i + 1] = aux;
         }
      }
   }
}

void insertionSort(int* v, int n) {
   int i, j, aux;
   for (i = 1; i < n; i++) {
      aux = v[i];
```

```
        j = i - 1;
        while (j >= 0 && v[j] > aux) {
            v[j + 1] = v[j];
            j = j - 1;
        }
        v[j + 1] = aux;
    }
}

void selectionSort(int* v, int n) {
    int i, j, min, aux;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (v[j] < v[min])
                min = j;
        }
        if (v[i] != v[min]) {
            aux = v[i];
            v[i] = v[min];
            v[min] = aux;
        }
    }
}

int ex3() {
    int dados[] = {1, 6, 8, 2, 4, 9, 5};
    int tam = sizeof(dados) / sizeof(dados[0]);

    printf("Dados originais: ");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i]);
    }
    printf("\n");

    bubbleSort(dados, tam);
    printf("Bubble Sort: ");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i]);
    }
    printf("\n");

    int dados2[] = {1, 6, 8, 2, 4, 9, 5};

    insertionSort(dados2, tam);
    printf("Insertion Sort: ");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados2[i]);
    }
    printf("\n");

    int dados3[] = {1, 6, 8, 2, 4, 9, 5};
```

```
    selectionSort(dados3, tam);
    printf("Selection Sort: ");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados3[i]);
    }
    printf("\n");

    printf("Soma dos elementos:\n");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i] + dados2[i] + dados3[i]);
    }
    printf("\n");

    printf("Diferença dos elementos:\n");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i] - dados2[i] - dados3[i]);
    }
    printf("\n");

    printf("Produto dos elementos:\n");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i] * dados2[i] * dados3[i]);
    }
    printf("\n");

    printf("Divisão dos elementos:\n");
    for (int i = 0; i < tam; i++) {
        printf("%d ", dados[i] / dados2[i] / dados3[i]);
    }
    printf("\n");

    return 0;
}
```

4 – Faça um teste de mesa com os algoritmos QuickSort e MergeSort, utilizando as seguintes sequências de dados de entrada:

a) V1[ ]={5,7,2,8,1,6}
b) V2[ ]={2,4,6,8,10,12,11,9,7,5,3,1}
c) V3[ ]={89,79,32,38,46,26,43,38,32,79}

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

```
void troca(int vet[], int a, int b) {
  int temp = vet[a];
  vet[a] = vet[b];
  vet[b] = temp;
}

int particiona(int vet[], int inicio, int fim) {
  int pivo = vet[fim];
  int pivo_indice = inicio;
  for (int i = inicio; i < fim; i++) {
    if (vet[i] <= pivo) {
      troca(vet, i, pivo_indice);
      pivo_indice++;
    }
  }
  troca(vet, pivo_indice, fim);
  return pivo_indice;
}

int particiona_random(int vet[], int inicio, int fim) {
  int pivo_indice = (rand() % (fim - inicio + 1)) + inicio;
  troca(vet, pivo_indice, fim);
  return particiona(vet, inicio, fim);
}

void quick_sort(int vet[], int inicio, int fim) {
  if (inicio < fim) {
    int pivo_indice = particiona_random(vet, inicio, fim);
    quick_sort(vet, inicio, pivo_indice - 1);
    quick_sort(vet, pivo_indice + 1, fim);
  }
}

void merge(int vet[], int left, int middle, int right) {
  int helper[right + 1];
  for (int i = left; i <= right; i++) {
    helper[i] = vet[i];
  }

  int i = left;
  int j = middle + 1;
  int k = left;
  while (i <= middle && j <= right) {
    if (helper[i] <= helper[j]) {
      vet[k] = helper[i];
      i++;
    } else {
      vet[k] = helper[j];
      j++;
    }
    k++;
  }
```

```
  while (i <= middle) {
    vet[k] = helper[i];
    i++;
    k++;
  }
}

void mergeSort(int vet[], int left, int right) {
  if (left < right) {
    int middle = (left + right) / 2;
    mergeSort(vet, left, middle);
    mergeSort(vet, middle + 1, right);
    merge(vet, left, middle, right);
  }
}

int ex4() {
  int V1[] = {5, 7, 2, 8, 1, 6};
  int V2[] = {2, 4, 6, 8, 10, 12, 11, 9, 7, 5, 3, 1};
  int V3[] = {89, 79, 32, 38, 46, 26, 43, 38, 32, 79};
  int tamanho_V1 = sizeof(V1) / sizeof(V1[0]);
  int tamanho_V2 = sizeof(V2) / sizeof(V2[0]);
  int tamanho_V3 = sizeof(V3) / sizeof(V3[0]);

  printf("Array V1 antes da ordenacao: ");
  for (int i = 0; i < tamanho_V1; i++) {
    printf("%d ", V1[i]);
  }
  printf("\n");

  printf("Array V2 antes da ordenacao: ");
  for (int i = 0; i < tamanho_V2; i++) {
    printf("%d ", V2[i]);
  }
  printf("\n");

  printf("Array V3 antes da ordenacao: ");
  for (int i = 0; i < tamanho_V3; i++) {
    printf("%d ", V3[i]);
  }
  printf("\n\n");

  quick_sort(V1, 0, tamanho_V1 - 1);
  quick_sort(V2, 0, tamanho_V2 - 1);
  quick_sort(V3, 0, tamanho_V3 - 1);

  printf("Sequência de Dados V1 depois do QuickSort: ");
  for (int i = 0; i < tamanho_V1; i++) {
    printf("%d ", V1[i]);
  }
  printf("\n");
```

```
  printf("Sequência de Dados V2 depois do QuickSort: ");
  for (int i = 0; i < tamanho_V2; i++) {
    printf("%d ", V2[i]);
  }
  printf("\n");

  printf("Sequência de Dados V3 depois do QuickSort: ");
  for (int i = 0; i < tamanho_V3; i++) {
    printf("%d ", V3[i]);
  }
  printf("\n\n");

  int V1_new[] = {5, 7, 2, 8, 1, 6};
  int V2_new[] = {2, 4, 6, 8, 10, 12, 11, 9, 7, 5, 3, 1};
  int V3_new[] = {89, 79, 32, 38, 46, 26, 43, 38, 32, 79};

  mergeSort(V1_new, 0, tamanho_V1 - 1);
  mergeSort(V2_new, 0, tamanho_V2 - 1);
  mergeSort(V3_new, 0, tamanho_V3 - 1);

  printf("Sequência de Dados V1 depois do MergeSort: ");
  for (int i = 0; i < tamanho_V1; i++) {
    printf("%d ", V1_new[i]);
  }
  printf("\n");

  printf("Sequência de Dados V2 depois do MergeSort: ");
  for (int i = 0; i < tamanho_V2; i++) {
    printf("%d ", V2_new[i]);
  }
  printf("\n");

  printf("Sequência de Dados V3 depois do MergeSort: ");
  for (int i = 0; i < tamanho_V3; i++) {
    printf("%d ", V3_new[i]);
  }
  printf("\n");

  return 0;
}


```

PONTEIROS E SUAS OPERAÇÕES

5 – Qual o valor de y no final do programa? Apresente a descrição do algoritmo linha a linha demonstrando o valor atribuído a cada variável durante a execução do algoritmo.

```c
#include<stdio.h>

int ex5() {
   int y, *p, x;
   y = 0; // Inicializa a variável y com o valor 0
   p = &y; // Atribui o endereço de y ao ponteiro p
   x = 4; // Atribui o valor 4 à variável x
   (*p)++; // Incrementa o valor apontado por p (y)
   x--; // Decrementa o valor de x
   (*p) += x; // Adiciona o valor de x ao valor apontado por p (y)
   printf("y = %d\n", y); // Imprime o valor final de y
   return 0;
}
```

6 – Com base nos conceitos aprendidos sobre ponteiros, analise o trecho de código escrito em linguagem C e classifique as sentenças abaixo em V (Verdadeira) ou F (Falsa):

```c
#include <stdio.h>
int main()
{
int a[] = {1,2,3,4,5);
int *pa;
pa=&a[0];
pa=a;
printf("%d",a[3]);
printf("%d",*(a+3));
printf("%d",pa[1]);
printf("%d",*(pa+1));
pa++;
a=pa;
a++;
return 0;
}
```

(F) A operação realizada na linha 8 é válida, o que torna os valores de pa e a idênticos, ou seja, apontam para o mesmo endereço de memória.
(F) A operação realizada na linha 8 é equivalente a operação realizada na linha 9, ou seja, produzem o mesmo resultado.
(V) A linha 10 e 11 ao serem executadas irão apresentar o mesmo resultado na tela para o usuário. Também pode-se afirmar que as linhas 12 e 13 são equivalentes e produzem o mesmo resultado.
(F) Na linha 14 encontra-se uma operação válida na manipulação de ponteiros na linguagem C.

(F) Ao analisar a linha 15 é possível afirmar que a = pa é uma operação válida na manipulação de ponteiros na linguagem C.
(F) A linha 16 apresenta uma operação válida na manipulação da estrutura de dados do tipo vetor.