2019/05/10 プログラミング演習

目的

この演習においてはモジュールの結合度・凝集度について触れる。

装置/ツール

- Visual Studio
- MacBook Pro

実験

問題4.1

> 2つの配列の共通集合を表示するvoid IntersectAndPrint(int [], int[])関数を実験書図4.2と図4.3の実行結果を元に作成し、ソースコードを示しなさい。

IntersectAndPrint関数を図4.1に示す。

```
static void IntersectAndPrint (int[] a, int[] b)
{
    int[] dup = new int[a.Length];

    int dupCnt = 0;

    for(int j = 0; j < a.Length; j++)
    {
        if (Containe(dup, dupCnt, a[j]))
        {
            continue;
        }

        for (int i = 0; i < b.Length; i++)
        {
            if (a[j] == b[i])
            {
                dup[dupCnt] = a[j];
                dupCnt++;
            }
        }
    }

    for (int i = 0; i < dupCnt; i++)
    {
        Console.Write(dup[i] + " ");
    }
}
```

図4.1 IntersectAndPrint関数

問題4.2

> 問題4.1で示されるIntersectAndPrint関数を機能的凝集度に照らし合わせて凝集度を高める。int[] intersect(int [], int[])関数とvoid Print(int[])関数に分割して実行するプログラムを新たに作成し、ソースコードを示しなさい。また実行結果をスク

> リーンショットで報告しなさい。

ソースコードを図4.2.1に示す。

```
using System;

namespace chapter4_1
{
    class Program
    {
        static bool Containe (int[] a, int aLen, int b)
        {
            for(int i = 0; i < aLen; i++)
            {
                if (a[i] == b)
                {
                    return true;
                }
            }
            return false;
        }

        static int[] Intersect(int[] a, int[] b)
        {
            int[] dup = new int[a.Length];

            int dupCnt = 0;

            for (int j = 0; j < a.Length; j++)
            {
                if (Containe(dup, dupCnt, a[j]))
                {
                    continue;
                }

                for (int i = 0; i < b.Length; i++)
                {
                    if (a[j] == b[i])
                    {
                        dup[dupCnt] = a[j];
                        dupCnt++;
                    }
                }
            }

            int[] ans = new int[dupCnt];
            for(int i = 0; i < dupCnt; i++)
            {
                ans[i] = dup[i];
            }

            return ans;
        }

        static void Print (int[] a)
        {
            for (int i = 0; i < a.Length; i++)
            {
                Console.Write(a[i] + " ");
            }
        }

        static void Main(string[] args)
        {
            int[] a1 = new int[5] { 1, 2, 3, 4, 5 };
            int[] a2 = new int[5] { 3, 1, 5, 10, 11 };

            Print(Intersect(a1, a2));

            Console.ReadKey();
        }
    }
```

```
}
```

図4.2.1 Print関数とIntersect関数に分割したソースコード

実行結果を図4.2.2に示す。

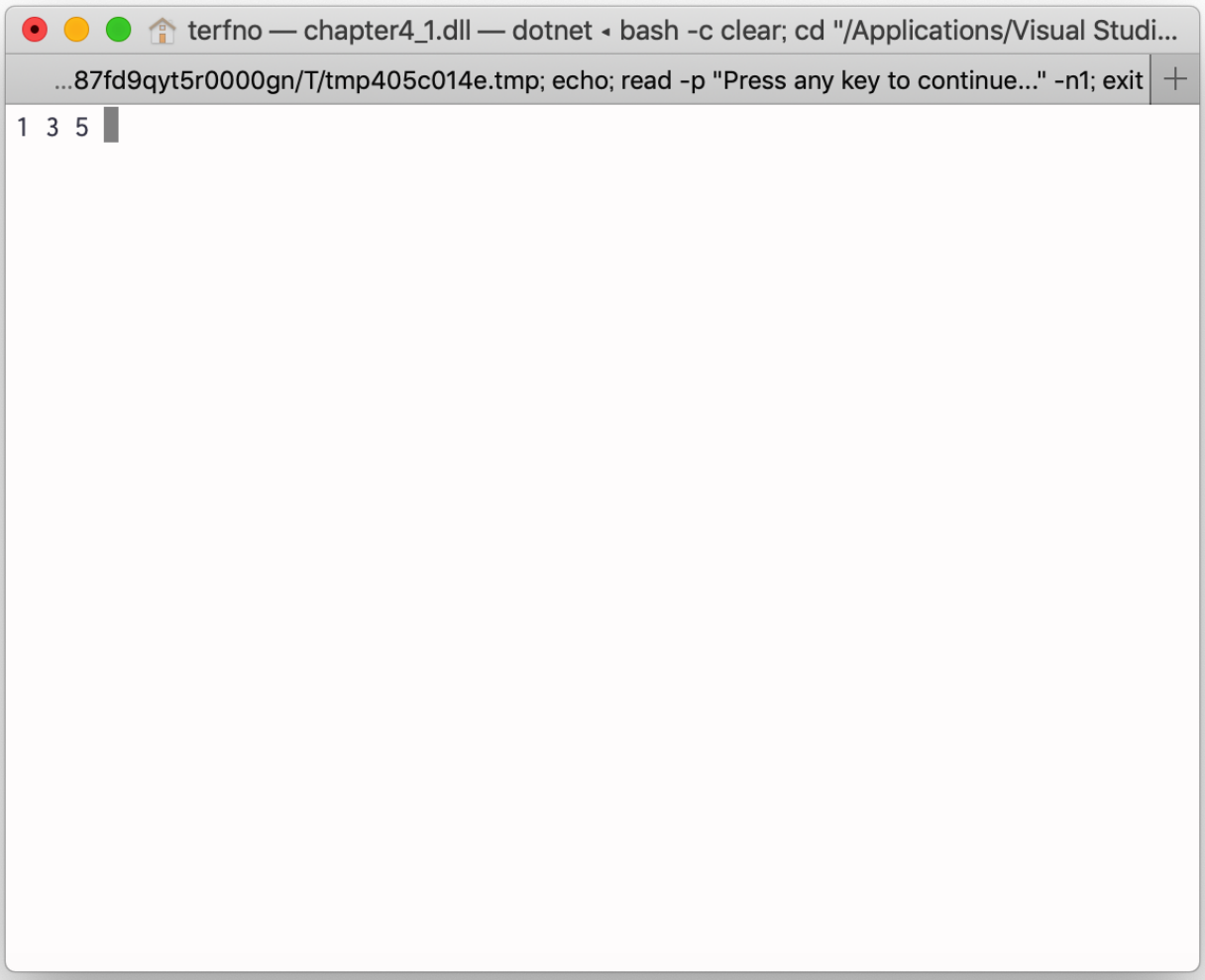

図4.2.2 実行結果のスクリーンショット

問題4.3

> 与えられたn×nの2次元配列を90度時計回りに回転させた回転行列を標準出力する関数void RotateAndPrint(int[,])を作成しソースコードを示しなさい。

作成したRotateAndPrint関数のソースコードを図4.3.1に示す。

```
static void RotateAndPrint(int[,] origin)
{
    int n = origin.GetLength(0);
    for(int i = 0; i < n; i++)
    {
        for(int j = n-1; j>=0; j--)
        {
            Console.Write(origin[j, i] + "\t");
        }
        Console.WriteLine();
    }
}
```

図4.3.1 RotateAndPrint関数

実行結果のスクリーンショットを図4.3.2に示す。

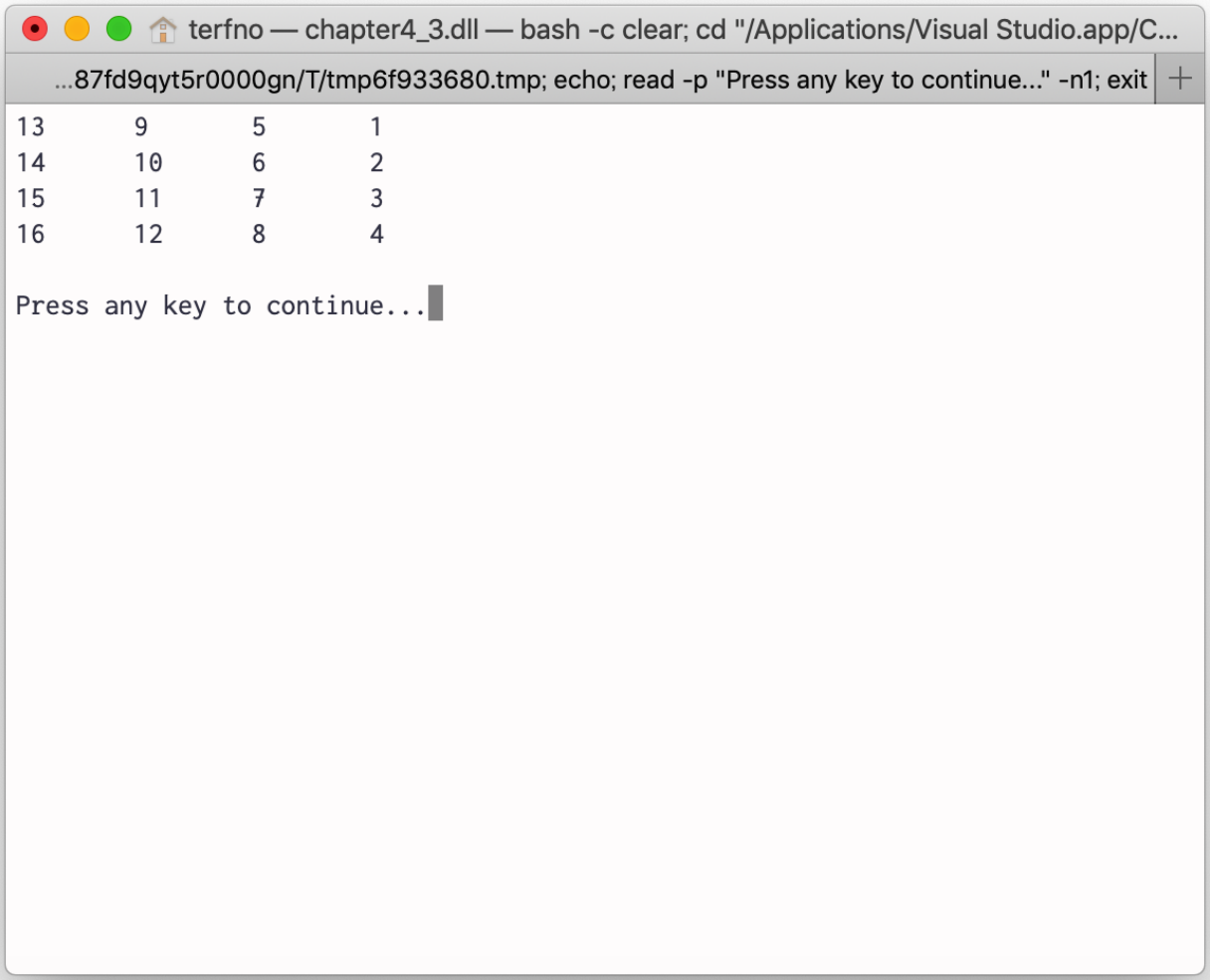

図4.3.2 4x4の2次元配列を$\frac{\pi}{2}$だけ回転させた実行結果

問題4.4

> 問題4.3の関数を機能的凝集度に照らし合わせて凝集度を高める。Rotate関数とPrint関数に分割し実行結果をスクリーンショットで示しなさい。

Rotate関数を図4.4.1に示す。

```
static int[,] Rotate(int[,] origin)
{
    int n = origin.GetLength(0);
    int[,] rotatedArray = new int[n, n];

    for (int i = 0; i < n; i++)
    {
        for (int j = n - 1; j >= 0; j--)
        {
            rotatedArray[i, n - 1 - j] = origin[j, i];
        }
    }

    return rotatedArray;
}
```

図4.4.1 Rotate関数

Print関数を図4.4.2に示す。

```
static void Print(int[,] array)
{
    for(int i = 0; i < array.GetLength(0); i++)
    {
        for(int j = 0; j < array.GetLength(0); j++)
        {
            Console.Write(array[i, j] + "\t");
        }
        Console.WriteLine();
    }
}
```

図4.4.2 Print関数

実行結果を図4.4.3に示す。

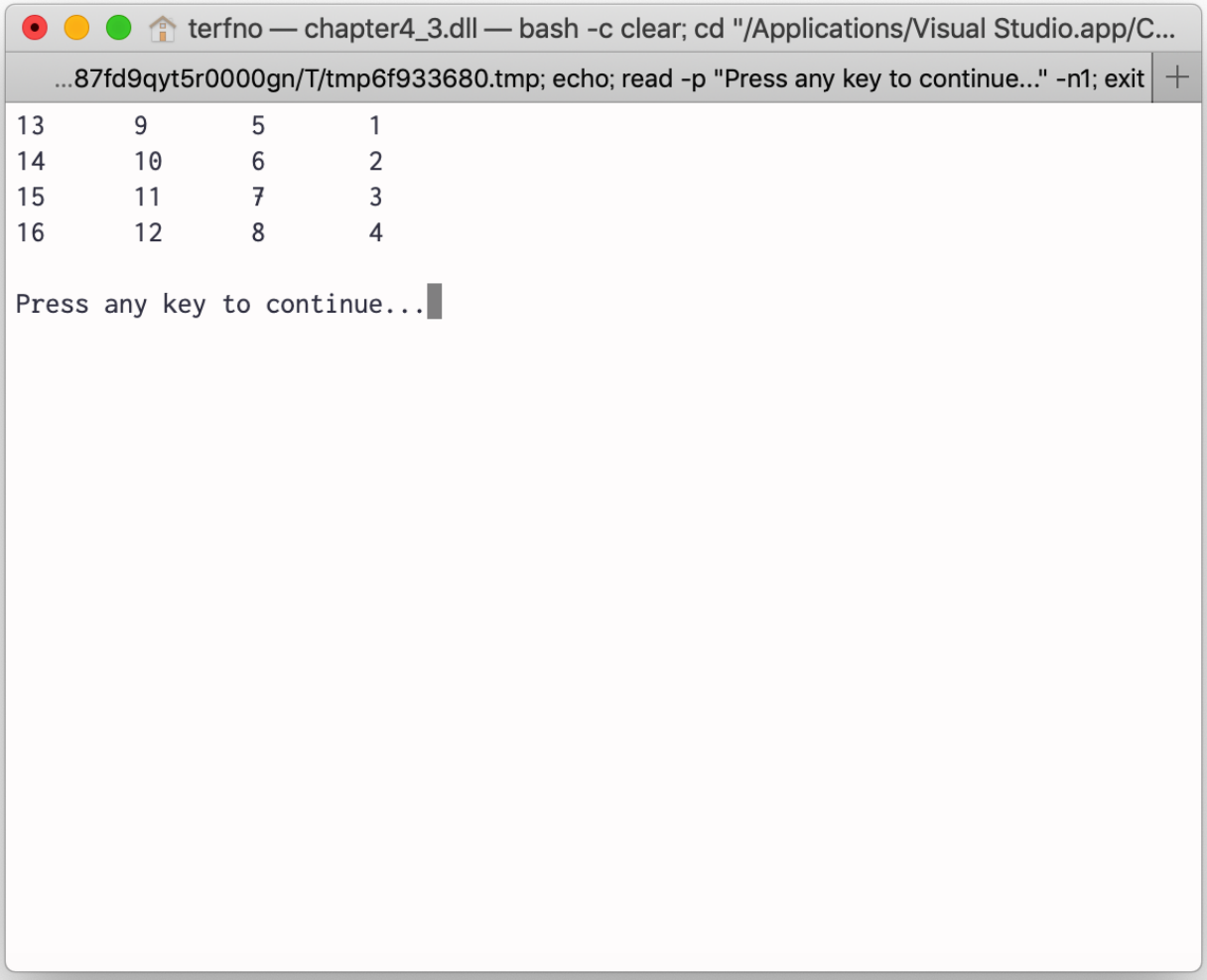

図4.4.3 4x4の2次元配列を$\frac{\pi}{2}$だけ回転させた実行結果

課題

課題4.1

> ソースコード中の2つの関数SwitchFlagとFuncArraysに対して、結合度の段階として何結合となっているか説明しなさい。
> また、結合度を下げるための提案をしなさい。

SwitchFlagがFuncArray内のflagを制御しているため、制御結合である。
結合度を下げる方法としてflagをグローバルではない変数として扱うことが考えられます。

課題4.2

> 実験書図4.1のソースコードで示されているFuncArrays関数の凝集度を段階別に示すと何凝集度になるか説明しなさい。

if文を用いて処理の選択がなされているため論理的凝集度である。

課題4.3

> 次の関数名が望ましくない理由を述べなさい。

> - HandleCalculation
> - OutputUser1, OutputUser2
> - Hoge

HandleCalculation

関数名からどのCalculationをハンドルするのかわからないため。

OutputUser1, OutputUser2

OutputUserで共通の処理が想定されるので、重複した処理を複数の関数で実行していてナンセンスなため。

Hoge

言わずと知れたメタネーミング。