TUẦN 4. DỮ LIỆU KIỂU MẢNG

1. Trong các câu lệnh khai báo và khởi tạo mảng dưới đây, câu nào sai

```
a. int  arr[ ] = new int[5];
b. int[ ]  arr = new int[5];
c. int  arr[ ] = new int[5];
d. int arr[ ]   new = int[5];
```

2. Giả sử trong chương trình có các câu lệnh dưới đây. Hãy cho biết khi thực hiện sẽ in ra kết quả như thế nào:

```
int arr[] = new int [5];
System.out.print(arr);
```

3. Mô tả và giải thích điều gì xảy ra khi biên dịch chương trình có dòng lệnh

```
int[] array = new int[-10];
```

4. Mô tả điều gì xảy ra khi bạn biên dịch chương trình chứa các dòng lệnh sau? Hãy giải thích.

```
int n = 1000;
int[] a = new int[n*n*n*n];
```

5. Mô tả điều gì sẽ xảy ra khi bạn biên dịch và thực hiện chương trình có các dòng lệnh sau. Hãy giải thích.

```
int[] a;
for (int i = 0; i < 10; i++)
      a[i] = i * i;
```

6. Khối lệnh sau nếu thực hiện sẽ in ra kết quả như thế nào? Hãy giải thích tại sao.

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
System.out.println(a==b);
```

7. Hãy cho biết điều gì xảy ra nếu trong chương trình EditArray.java (trong bài giảng lý thuyết tuần thứ 4), ta thay mã chương trình phương thức modifyArray thành như sau:

```
public static void modifyArray(int array[]){
      for(int number:array)
            number *= 2.0;
      /* so với đoạn mã gốc:
      for (int i = 0; i<array.length; i++)
            array[i] *= 2.0;      *******/
}
```

8. Cho b là một mảng 100 phần tử với giá trị ban đầu là 0, và a là mảng N phần tử với các phần tử là số nguyên trong khoảng từ 0 đến 99. Hãy cho biết vòng lặp sau thực hiện việc gì?

```
for (int j = 0; j < N; j++)
      b[a[j]] ++;
```

9. Điều gì xảy ra khi biên dịch và thực hiện (nếu được) chương trình sau. Hãy giải thích.

```
public class Test{
      public static void main(String args[]){
            int x[] = {1, 2, 3, 4};
            int   y  = x;
            x = new int[2];
            for(int  i = 0; i < x.length; i++)
                  System.out.print(y[i] + " ");
      }
}
```

Trong các bài tập dưới đây, nếu có yêu cầu viết chương trình, bạn hãy thực hiện việc viết dưới dạng chương trình con. Hàm main chỉ phục vụ thông báo, nhập dữ liệu và gọi các chương trình con.

Nếu không chú thích thêm, thì dữ liệu (*các phần tử*) *mảng được nhập thông qua tệp văn bản. Dữ liệu khác* (*kích thước mảng, giá trị cần kiểm tra*) *được nhập qua đối dòng lệnh*

1. Xây dựng lớp ProcessArray, trong đó có hàm countValue(double x, double Array[]) đếm xem giá trị x xuất hiện bao nhiêu lần trong mảng Array, trả lại số lần xuất hiện. Hàm main cho phép nhận một số thực x (double), một số nguyên n (int) và một mảng n phần tử thực (double). Tệp đầu vào có quy tắc: dòng đầu ghi số thực x; dòng tiếp theo ghi số nguyên n; dòng cuối ghi n số thực của mảng, giữa các số có 01 khoảng trống.
2. Bổ sung vào lớp ProcessArray ở bài trước hàm minArray(double Array[]), trả về giá trị nhỏ nhất; hàm maxArray(double Array[]), trả về giá trị lớn nhất của mảng Array kiểu double. Bổ sung vào hàm main của bài trước dòng lệnh in ra giá trị phần tử nhỏ nhất và lớn nhất của mảng.
3. Bổ sung vào lớp ProcessArray ở bài trước hàm indexValue(double x, double Array[]), kiểm tra nếu giá trị x xuất hiện trong mảng Array kiểu double, trả lại mảng số nguyên là chỉ số các phần tử có giá trị x. Bổ sung vào hàm main của bài 2 dòng gọi hàm indexValue và in ra màn hình nhưng vị trí xuất hiện của x, trên cùng dòng cách nhau bởi khoảng trắng.
4. Bổ sung vào lớp ProcessArray ở bài trước hàm/phương thức sortIncArray(double Array[]), sắp xếp các phần tử mảng theo thứ tự tăng dần về giá trị; hàm/phương thức sortDecArray(double Array[]), sắp xếp các phần tử mảng theo thứ tự giảm dần về giá trị,

các chương trình còn này trả lại các mảng đã được sắp xếp. Bổ sung vào hàm main của bài 3 dòng gọi sortIncArray và sortDecArray in ra màn hình các mảng kết quả sau khi đã sắp xếp.

5. Xây dựng lớp Vector, trong đó có các hàm/phương thức sau
    a. VectorNorm(double vector[]), trả về chuẩn 2 của vector có các thành phần chứa trong mảng số thực vector. Công thức tính chuẩn vector $a = (a_i)$ như sau

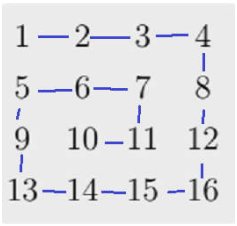
$$\|(a_1, a_2 \ldots, a_n)\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

    b. ScalarProd(double vector1[], double vector2[]) trả lại tích vô hướng của hai vector có thành phần trong hai mảng tương ứng. Công thức tính tích vô hướng của hai vector $a = (a_i)$ và $b = (b_i)$ như sau

$$(a, b) = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n.$$

    c. IsOrthogonal(double vector1[], double vector2[]) kiểm tra xem hai vector tham đối có trực giao với nhau hay không, trả về true hoặc false. Các vector là trực giao nếu tích vô hướng của chúng bằng 0.
    d. sumVectors(double vector1[], double vector2[]) trả về vector tổng của hai vector tham đối. Tổng $c = (c_i)$ của các vector $a = (a_i)$ và $b = (b_i)$ có $c_i = a_i + b_i$.
    e. Phương thức main cho phép nhập hai số nguyên m và n, sau đó nhập các mảng số thực có tương ứng m và n phần tử. Gọi hàm tính vào in ra chuẩn 2 của các vector ứng với 2 mảng vừa nhập. Kiểm tra số phần tử có bằng nhau hay không, nếu bằng nhau hãy in ra vector tổng của chúng. Sau đó kiểm tra xem chúng có trực giao hay không. Nếu chúng không trực giao, hãy in ra tích vô hướng của chúng.
6. Cho ma trận số nguyên `A[][]` kích thước $m \times n$. Xây dựng lớp intMatrix, có các phương thức sau, với tham đối là mảng `A[][]` tương ứng:
    a. printMatrix(int A[][]) in ma trận ra màn hình theo dạng giống hàng cột, với chỉ số thứ nhất là chỉ số hàng, chỉ số thứ hai là chỉ số cột.
    b. setZeros(int A[][]): Duyệt ma trận xem phần tử `A[i][j]` có bằng 0 hay không. Nếu nó bằng không, hay thay giá trị các phần tử của hàng *i* và cột *j* thành 0 (*Chú ý*: *không được sử dụng thêm mảng phụ nào khác*).
    c. findMiniMax(int A[][]): Tìm xem có phần tử `A[i][j]` nào thỏa mãn tính chất: có giá trị bé nhất trong hàng *i* nhưng là lớn nhất trong cột *j* – còn gọi là điểm yên ngựa - hay không. Nếu có, hãy trả về một mảng 2 chiều lưu các cặp chỉ số (i, j) là vị trí xuất hiện điểm yên ngựa.

d. printSpiralVector(int A[][]): In các giá trị phần tử mảng theo đường xoáy ốc thành một dãy. Ví dụ với ma trận sau thì dãy kết quả cần in ra sẽ là



1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

e. Hàm main() cho phép nhận các kích thước m, n qua đối dòng lệnh, khởi tạo sau đó sinh ngẫu nhiên các phần tử mảng A[][] kích thước m×n, giá trị nguyên trong khoảng [-100, 100]. Tiếp theo thực hiện các công việc sau:

   i. In ma trận A ra màn hình theo dạng dòng cột;

   ii. Gọi phương thức setZeros để thực hiện với ma trận A, sau đó in lại ma trận kết quả theo dạng dòng cột.

   iii. Tìm vị trí các điểm yên ngựa của ma trận A, nếu có;

   iv. Áp dụng phương thức printSpiralVector đối với A.

7. Ma trận vuông A kích thước n × n được gọi là ma trận ma thuật (màu nhiệm – magic matrix) nếu nó có các tổng của phần tử trên đường chéo chính, trên các hàng và trên các cột là bằng nhau. Hãy lập một lớp MagicMatrix có hàm isMagicMatrix(int A[][]) kiểm tra xem A có phải là ma trận ma thuật hay không, nếu đúng trả lại true, ngược lại là false. Ngoài ra còn có hàm sumDiag(int A[][]) tính và trả về tổng các phần tử trên mỗi đường chéo chính của ma trận A. Viết hàm main nhận số nguyên n từ đối dòng lệnh, sau đó sinh ngẫu nhiên ma trận A vuông kích thước n, các phần tử nguyên trong khoảng [-100, 100]. Sau đó tính và in ra tổng các phần tử trên đường chéo chính của A và kiểm tra xem A có là ma trận ma thuật hay không.

8. Ma trận vuông 9 × 9 có giá trị các phần tử là các số nguyên từ 1 đến 9, được gọi là nghiệm đúng trò chơi Sudoku nếu mỗi hàng, mỗi cột, mỗi khối ma trận 3×3 trong 9 khối con tạo thành khối ma trận 9 × 9 ban đầu, đều chứa đủ các số từ 1 đến 9, như hình minh họa bên phải. Hãy viết chương trình sinh ngẫu nhiên một mảng 2 chiều 9 × 9, in ma trận ra màn hình dạng gióng dòng cột, sau đó kiểm tra xem nó có tạo thành một ma trận nghiệm đúng trò chơi Sudoku hay không? Phần kiểm tra và phần in ma trận cần được viết thành các chương trình con.

```
 5 3 4 | 6 7 8 | 9 1 2
 6 7 2 | 1 9 5 | 3 4 8
 1 9 8 | 3 4 2 | 5 6 7
-------+-------+------
 8 5 9 | 7 6 1 | 4 2 3
 4 2 6 | 8 5 3 | 7 9 1
 7 1 3 | 9 2 4 | 8 5 6
-------+-------+------
 9 6 1 | 5 3 7 | 2 8 4
 2 8 7 | 4 1 9 | 6 3 5
 3 4 5 | 2 8 6 | 1 7 9
```