AMA2222 Principles of Programming

Leung Man Kin, Adam
Instructor

adam.leung@polyu.edu.hk
TU720

Chapter 6: Data Structure II - array
array, array processing, range-based for in an array
passing arrays to functions, two dimensional arrays

Array

An array is used to store multiple values of the same type.
An element in an array can be accessed using an index.
To declare an array, specify its element type and size which must be a constant.

Three programmers walk into a bar...

Syntax for declaring an array:

```
dataType arrayName[size];
```

Syntax for calling one element of the array:

```
arrayName[index]
```

Notice that `size` represents total number of elements of the array, where the index must be between 0 to `(size-1)`.
In older version of C++, the size has to be a `const`.

An array can also be initialized either by each element or using the following syntax:

```
elementType arrayName[size]={value0, value1,
... , valuek};
```

For example, the following array contains 4 elements

```
double myList[4] = {1.9, 2.8, 3.4, 7.2};
```

You may also write:

```
double myList[4];
myList[0] = 1.9;
myList[1] = 2.8;
myList[2] = 3.4;
myList[3] = 7.2;
```

Example program 6.1 Array of numbers and their sum

```
1  // 6.1 array of numbers
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6          const int size = 10;
7          double numbers[size];
8          double sum = 0;
9
10         for (int i=0; i<size; i++)
11         {
12                 cout << "Enter a new number: ";
13                 cin >> numbers[i];
14                 sum += numbers[i];
15         }
16
17         double average = sum/size;
18         cout << "Average is " << average << endl;
19         return 0;
20 }
```

Declaring an array with a constant number of elements. This constant is set as 10 but can be changed.

The for-loop will go as the index i increases from 0 to 9. In each iteration, a number is read to `numbers[i]`.

Example program 6.2 Fibonacci sequence stored in array

```
1  // 6.2 Fibonacci sequence
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6           const int size = 20;
7           int fib[size]={1, 1};
8           cout << "The first " << size <<
9  " Fibonacci numbers are: " << endl;
10          cout << fib[0] << " ";
11          cout << fib[1] << " ";
12          for (int i=2; i<size; i++)
13          {
14                    fib[i] = fib[i-1] + fib[i-2];
15                    cout << fib[i] << " ";
16          }
17          return 0;
18 }
```

Declaring an array with a constant number of elements.
Only initialize the first two elements.

Starting from fib[2], each element is the sum of the two consecutive elements before.

The output is:

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

Processing arrays

We have learnt how to define an array. Next we will learn some common techniques in processing the array.

Initializing arrays with (console) input values

```
datatype myList[arraySize];
cout << "Enter " << arraySize << " values ";
for (int i = 0; i < size; i++)
{
          cin >> myList[i];
}
```

Initializing arrays with (file) input values

```
datatype myList[arraySize];
file.open("file_name.txt")
for (int i = 0; i < size; i++)
{
          file >> myList[i];
}
```

Initializing arrays with zeros

```
datatype myList[arraySize] = {};
```

Printing arrays

```
for (int i = 0; i < arraySize; i++)
{
        cout << myList[i] << " ";
}
```

Writing arrays to file

```
file.open("file_name.txt")
for (int i = 0; i < arraySize; i++)
{
        file << myList[i] << " ";
}
```

Copying arrays

```
datatype newList[arraySize];
for (int i = 0; i < arraySize; i++)
{
        newList[i] = myList[i];
}
```

You might be thinking of **`newList = myList`** but this is not allowed in C++! Need to process each element one-by-one.

Summing all elements

```
double total = 0;
for (int i = 0; i < arraySize; i++)
{
          total += myList[i];
}
```

Counting number of elements satisfying a condition

```
int count = 0;
for (int i = 0; i < arraySize; i++)
{
          if (boolean logical condition)
                    count++;
}
```

Finding the largest element and its index

```
double max = myList[0];
int index_max = 0;
for (int i = 1; i < arraySize; i++)
{
          if (myList[i] > max)
          {
          max = myList[i];
          index_max = i;
          }
}
```

We first assume that the first element myList [0] is the largest element. When we move the next element, if it is larger than the maximum so far, update the maximum value and the index.

Finding the smallest element and its index

```
double min = myList[0];
int index_min = 0;
for (int i = 1; i < arraySize; i++)
{
          if (myList[i] < min)
          {
          min = myList[i];
          index_min = i;
          }
}
```

For finding smallest value, we just change max into min and larger to smaller.

Example Program 6.3

midterm.txt contains the marks of 40 students. Read and store them as an array of integers. Then evaluate the following:

Maximum mark
Minimum mark
Mean mark
Number of fails (below 40)

```
19 81 45 71 64 76 43 76 58 40
76 91 41 82 68 50 41 21 42 58
80 75 48 45 24 75 89 86 19 82
50 35 22 80 29 72 69 77 88 68
```

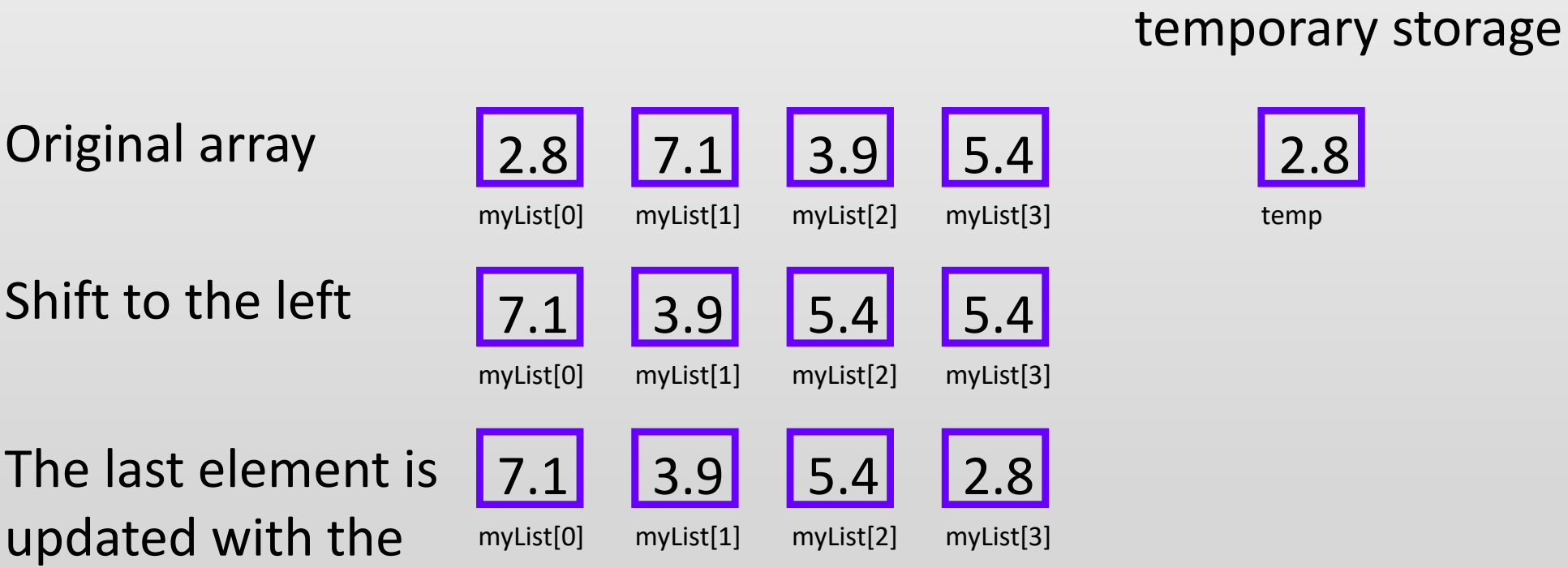
```
 1 // 6.3 midterm marks
 2 #include <iostream>
 3 #include <fstream>
 4 using namespace std;
 5 int main()
 6 {
 7          ifstream file;
 8          file.open("midterm.txt");
 9          int mark[40];
10          for (int i=0; i<40; i++)
11                    file >> mark[i];
12          file.close();
```

```
13          double sum = 0, max = 0, min = 100;
14          int count = 0;
15          for (int i=0; i<40; i++)
16          {
17                  sum += mark[i];
18                  if (mark[i] > max)
19                          max = mark[i];
20                  if (mark[i] < min)
21                          min = mark[i];
22                  if (mark[i] < 40)
23                          count++;
24          }
25          double mean = sum/40;
26          cout << "Maximum mark is " << max << endl;
27          cout << "Minimum mark is " << min << endl;
28          cout << "Mean mark is " << mean << endl;
29          cout << "Number of fails is " << count << endl;
30          return 0;
31  }
```

```
Maximum mark is 91
Minimum mark is 19
Mean mark is 58.9
Number of fails is 7
```

Shifting elements

```
double temp = myList[0];
for (int i = 1; i < arraySize; i++)
{
        myList[i-1] = myList[i];
}
myList[arraySize-1] = temp;
```



Range-based for in an array

In array processing, we often need to access each element in an array. This can be achieved by a one-dimensional for-loop, using integer `i` as index to access each element `list[i]`.

In C++11 onwards, with the **`auto`** data type we can directly access each element within the array. Compare the syntax:

```
for (int i; i<size; i++)
   { … list[i] … }

for (auto x: list)
   { … x … }
```

Example program 6.4 Accessing each element in array

```
 1 // 6.4a Accessing each element in array
 2 #include <iostream>
 3 using namespace std;
 4 int main()
 5 {
 6           int prime[5] = {2, 3, 5, 7, 11};
 7           for (int i=0; i<5; i++)
 8                     cout << prime[i] << " ";
 9           return 0;
10 }
```

```
 1 // 6.4b Accessing each element in array
 2 #include <iostream>
 3 using namespace std;
 4 int main()
 5 {
 6           int prime[5] = {2, 3, 5, 7, 11};
 7           for (auto x: prime)
 8                     cout << x << " ";
 9           return 0;
10 }
```

The two programs show the same result. In 6.4a we go over all indexes while in 6.4b we go over all elements of the array.

Passing arrays to functions

When an array argument is passed to a function, its starting address is passed to the array parameter in the function. Both parameter and argument refer to the same array. If you change the array in the function, you will see the change outside the function.

Notice that this is different from passing a variable to a function. The change of value in local variable does not affect the global one.

Classwork 6.1: Evaluate the output of the following program:

```
#include <iostream>
using namespace std;
void m(int x, int y[])
{
        x = 3;
        y[0] = 3;
}
int main()
{
        int number = 0;
        int array[1];
        m(number, array);
        cout << "The number is " << number << endl;
        cout << "The array is " << array[0] << endl;
        return 0;
}
```

The number is 0
The array is 3

Many of you might think the function will assign value 3 to both quantities number and array[0].

Some of you might think both quantities will not be changed by the function.

In C++, a variable input to a function using pass by reference will not be updated. However, an array input to a function is regarded as pass by reference, therefore its array elements will be updated.

Example program 6.5 Passing array to function

```
1  // 6.5 Passing array to function
2  #include <iostream>
3  using namespace std;
4
5  void printArray(int list[], int size);
6
7  int main()
8  {
9          int numbers[5] = {1, 4, 3, 6, 8};
10         printArray(numbers, 5);
11         return 0;
12 }
13
14 void printArray(int list[], int size)
15 {
16         for (int i = 0; i < size; i++)
17         {
18                 cout << list[i] << " ";
19         }
20 }
```

Function prototype

Invoke the function, use the array and its size as input parameters

Function implementation

Example program 6.5 Reversing an array

```
 1 // 6.5 Reversing an array
 2 #include <iostream>
 3 using namespace std;
 4 void reverse(int list[], int size)
 5 {
 6           for (int i = 0; i < size/2; i++) {
 7                     int temp = list[i];
 8                     list[i] = list[size-1-i];
 9                     list[size-1-i] = temp; }
10 }
11 void printArray(int list[], int size)
12 {
13           for (int i = 0; i < size; i++)
14                     cout << list[i] << " ";
15 }
```

Function reverse reads an arrays its size, and then rearrange its elements in an reversed order.

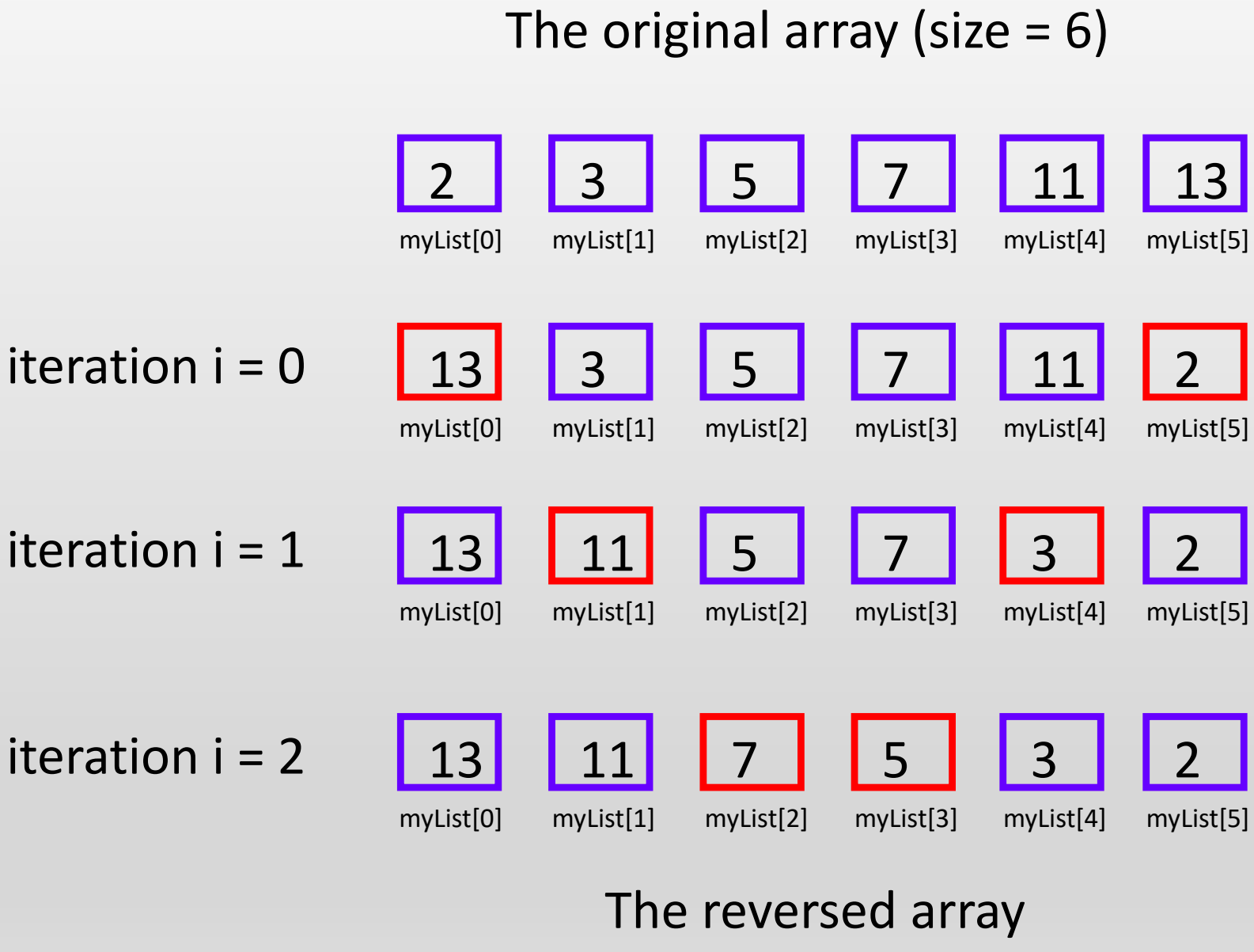
Function printArray reads an array and its size, and then print out all the elements in order.

```
16 int main()
17 {
18          int size = 6;
19          int list[] = {2, 3, 5, 7, 11, 13};
20          cout << "The original array: ";
21          printArray(list, size);
22          cout << endl;
23          reverse(list, size);
24          cout << "The reversed array: ";
25          printArray(list, size);
26          cout << endl;
27          return 0;
28 }
```

output:

```
The original array: 2 3 5 7 11 13
The reversed array: 13 11 7 5 3 2
```

In the main program, the array list is initialized. The function reverse is reverse the ordering of elements in list. Both the original list and the reversed one are printed for comparison.


The original array (size = 6)
2
3
5
7
11
13
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
iteration i = 0
13
3
5
7
11
2
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
iteration i = 1
13
11
5
7
3
2
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
iteration i = 2
13
11
7
5
3
2
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
The reversed array

Classwork 6.2

Write a function that reads an double array and its size, then return the maximum value. The main program has been given to you. The result should be 5.9.

```
 1 // cw6.2 maximum value of array
 2 #include <iostream>
 3 #include <fstream>
 4 using namespace std;
 5 //missing function
 6 int main()
 7 {
 8           double list[5] = {1.8, -3.4, 5.9, 2.6};
 9           cout << maxv(list, 5);
10           return 0;
11 }
```

Two-dimensional arrays

The kind of array we have learnt is one dimensional, which means its elements can be indicated by a single index. However, some quantities like matrix are expressed in two indexes (row number and column number). This can be represented by a two-dimensional array with two indexes.

The syntax for declaring a two-dimensional array is

```
elementType arrayName[ROW_SIZE][COLUMN_SIZE];
```

In general, an N-dimensional array can be declared by

```
elementType arrayName[size1][size2]…[sizeN];
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

For example, a is a 3X4 matrix. To assign the value 7 to a specific element at row 2 and column 1, use the following:

```
int a[3][4];
a[0][0] = 1;
a[0][1] = 2;
……
```

You can use an array initializer to declare and initialize a two-dimensional array. For example,

```
int a[3][4] = {{1,2,3,4},
               {5,6,7,8},
               {9,10,11,12}};
```

Function with multi-dimensional array as input

When passing a two-dimensional array to a function, C++ requires that the column size be specified in the function parameter type. For example, to pass an array of size 3×4 to a function, the function parameter should be defined as:

```
int functionName(int a[3][4])
{
    // function body statements
}
```

However, in calling the function there is no need to add the [][] signs. For example:

```
functionName(a);
```

Example program 6.7 Inputting a matrix

```
1  // 6.7 Input matrix
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6          int rowSize;
7          int colSize;
8          cout << "Enter number of rows: ";
9          cin >> rowSize;
10         cout << "Enter number of columns: ";
11         cin >> colSize;
12
13         int a[rowSize][colSize];
14         cout << "Enter the entries of the matrix: " << endl;
15         for (int i = 0; i < rowSize; i++)
16                 for (int j = 0; j < colSize; j++)
17                         cin >> a[i][j];
18         for (int i = 0; i < rowSize; i++)
19         {
20                 for (int j = 0; j < colSize; j++)
21                         cout << a[i][j] << " ";
22                 cout << endl;
23         }
24         return 0;
25 }
```

Read the size of the matrix

Reading the matrix entries using a two-dimensional for-loop

Printing the matrix using a two-dimensional for-loop

Processing two-dimensional arrays

Similar to the processing of one-dimensional array, but we need to consider a two-dimensional for-loop. For examples:

Summing all elements

```
double total = 0;
for (int i = 0; i < rowSize; i++)
        for (int j = 0; j < colSize; j++)
                total += myList[i][j];
```

However, C++ doesn't allow passing an entire two-dimensional array into a function as input. The number of columns have to be specified

```
double total = 0;
for (int i = 0; i < rowSize; i++)
        for (int j = 0; j < colSize; j++)
                total += myList[i][j];
```

However, C++ doesn't allow passing an entire two-dimensional array into a function as input. The number of columns have to be specified.

both row and column sizes specified

```
void printmatrix(int myList[2][3]){
for (int i = 0; i < 2; i++){
        for (int j = 0; j < 3; j++)
                cout << myList[i][j] << " ";
        cout << endl;}
}
```

only column size specified, row size depends on input parameter

```
void printmatrix(int myList[][3], int rowSize){
for (int i = 0; i < rowSize; i++){
        for (int j = 0; j < 3; j++)
                cout << myList[i][j] << " ";
        cout << endl;}
}
```

```
void printmatrix(int myList[][], int rowSize, int rowSize){
for (int i = 0; i < rowSize; i++){
        for (int j = 0; j < colSize; j++)
                cout << myList[i][j] << " ";
        cout << endl;}
}
```

both row and column sizes depend on input parameter (not allowed)

Example program 6.8 Determinant of a 2X2 matrix

```
 1 // 6.8 Determinant
 2 #include <iostream>
 3 using namespace std;
 4 int det(int m[2][2])
 5 {
 6           int d = m[0][0]*m[1][1]-m[1][0]*m[0][1];
 7           return d;
 8 }
 9 int main()
10 {
11           int a[2][2];
12           cout << "Enter the entries of a 2x2 matrix: " << endl;
13           for (int i = 0; i < 2; i++)
14                     for (int j = 0; j < 2; j++)
15                               cin >> a[i][j];
16           cout << "The determinant is " << det(a);
17           return 0;
18 }
```

Example program 6.9 Crammer's rule

In linear algebra, solving a system of n linear equations with n unknowns can be done by Crammer's rule. Suppose:

$$A\vec{x} = \vec{b} \quad \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

Then we have:

$$x_0 = \frac{\det \begin{pmatrix} b_0 & a_{01} \\ b_1 & a_{11} \end{pmatrix}}{\det \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}} \qquad x_1 = \frac{\det \begin{pmatrix} a_{00} & b_0 \\ a_{10} & b_1 \end{pmatrix}}{\det \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}}$$

Assuming that the determinant of A is non-zero. Write a program that prompt user to enter A and b then evaluate x.

```cpp
// 6.9 Crammer's rule
#include <iostream>
using namespace std;
double det(double m[2][2])
{
	double d = m[0][0]*m[1][1]-m[1][0]*m[0][1];
	return d;
}
int main()
{
	double a[2][2];
	double b[2];
	double x[2];
	cout << "Enter the entries of a 2x2 matrix: " << endl;
	for (int i = 0; i < 2; i++)
		for (int j = 0; j < 2; j++)
			cin >> a[i][j];
	cout << "Enter the entries of a 2x1 vector: " << endl;
	for (int i = 0; i < 2; i++)
		cin >> b[i];
	double dx[2][2] = {{b[0],a[0][1]},{b[1],a[1][1]}};
	double dy[2][2] = {{a[0][0],b[0]},{a[1][0],b[1]}};
```

```
23          if (det(a)!=0)
24          {         x[0] = det(dx)/det(a);
25                    x[1] = det(dy)/det(a);
26          cout << "The solution of the following system" << endl;
27          cout << a[0][0] << " x + " << a[0][1] << " y = " << b[0] << endl;
28          cout << a[1][0] << " x + " << a[1][1] << " y = " << b[1] << endl;
29          cout << "can be evaluate by Crammer's rule" << endl;
30          cout << "x = " << x[0] << endl;
31          cout << "y = " << x[1] << endl;
32          }
33          else cout << "Sorry, the determinant is 0";
34          return 0;
35  }
```

Example program 6.10 Symmetric matrix

The transpose of an $m \times n$ matrix can be obtained by flipping it along the diagonal. The result is an $n \times m$ matrix. For example:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

A matrix is symmetric if it equals to its own transpose. This implies the matrix must be a square matrix. For example:

$$B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix} = B^T$$

Write a function that reads an two dimensional array and check if it represents a symmetric matrix.

```
1  // 6.10 Symmetric matrix
2  #include <iostream>
3  using namespace std;
4  const int n = 3;
5  bool sym(int a[n][n])
6  {
7          for (int i = 0; i < n; i++)
8                  for (int j = 0; j < n; j++)
9                          if (a[i][j]!=a[j][i])
10                                 return false;
11         return true;
12 }
13 int main()
14 {
15         int m[n][n];
16         cout << "Enter the entries of the matrix: " << endl;
17         for (int i = 0; i < n; i++)
18                 for (int j = 0; j < n; j++)
19                         cin >> m[i][j];
20         if (sym(m)) cout << "It is a symmetric matrix." << endl;
21         else cout << "It is not a symmetric matrix." << endl;
22         return 0;
23 }
```

Define the size of the nxn square matrix

Check if any two pair of a_{ij} and a_{ji} are unequal, violating the symmetric property.

If all pairs of a_{ij} and a_{ji} are equal, the matrix is symmetric.

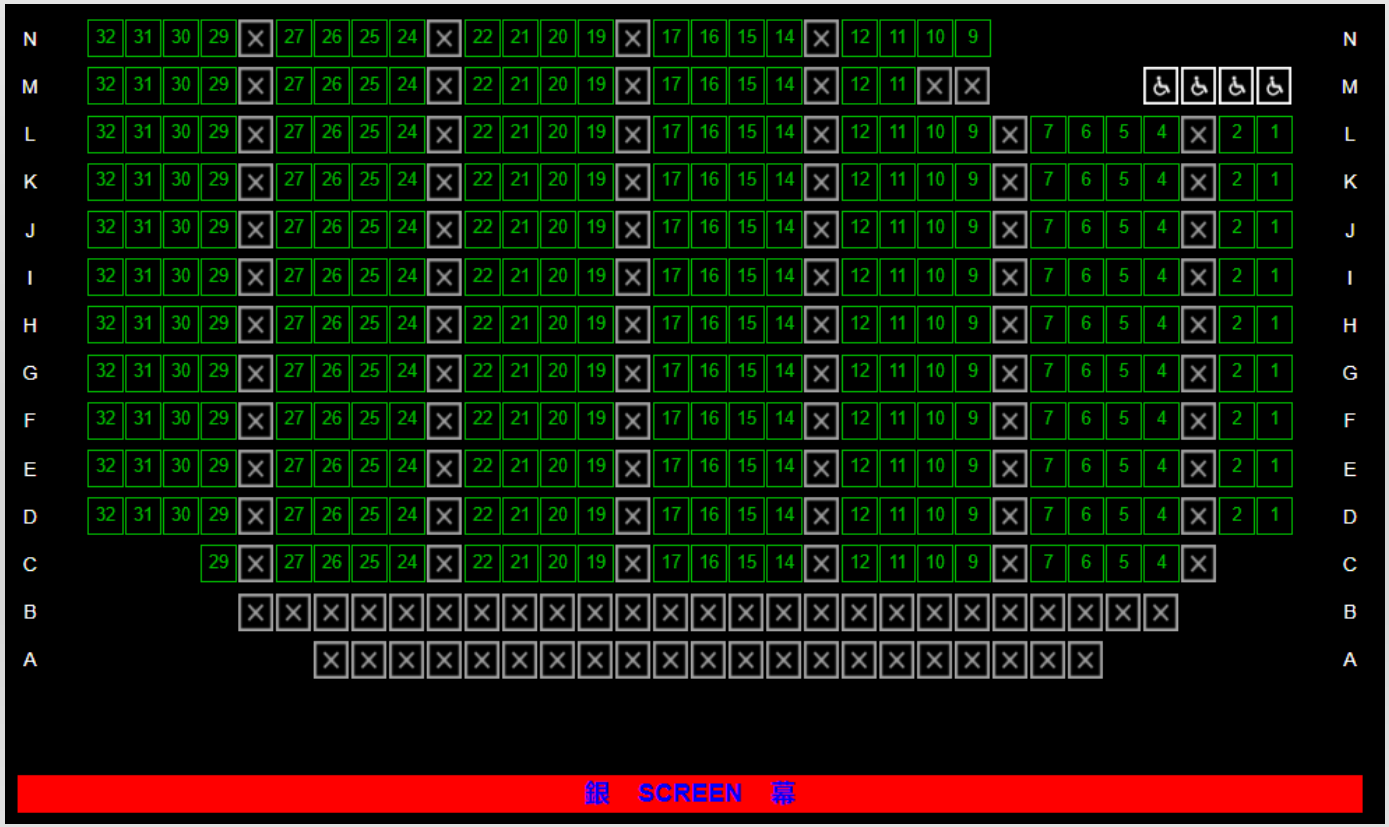
Read the nxn matrix.

Use the function to check if it is symmetric, print the result.

Example program 6.11 Ticketing system

A seating plan in a cinema can be considered as a two-dimensional array. The availability of each seat is a Boolean parameter.

Write a program that allows the user (teller at the ticketing booth) to check and update the availability of the seats upon selling tickets.



```
1  //6.11 ticketing
2  #include<iostream>
3  #include<iomanip>
4  using namespace std;
5
6  const int rn=10, cn=20;
7
8  void clearSeat(bool a[rn][cn])
9  {
10         for (int i=0; i<rn; i++)
11                 for (int j=0; j<cn; j++)
12                         a[i][j] = true;
13 }
14
15 void printSeat(bool a[rn][cn])
16 {
17         for (int i=0; i<rn; i++)
18         {
19                 cout << setw(3)<< (char)('A'+i);
20                 for (int j=0; j<cn; j++)
21                         if (a[i][j])
22                                 cout << setw(3) << j;
23                         else
24                                 cout << setw(3) << "X";
25                 cout << endl;
26         }
27 }
```

make all seats available

print number of available seats, mark 'X' for those unavailable

```
28 void buySeat(bool a[rn][cn], char letter, int number)
29 {
30         if (a[letter-'A'][number])  {
31                 cout << "You have purchased " << letter << number << endl;
32                 a[letter-'A'][number] = false;}
33         else cout << "Sorry! " << letter << number << " is not available" << endl;
34 }
35 int main()
36 {
37         bool movie1[rn][cn];
38         clearSeat(movie1);
39         char key = 'A';
40         cout << "P: print seating plan / C: clear the seating plan" << endl;
41         cout << "B: buy a seat / E: exit" << endl;
42         do {
43                 cout << "Please enter the instruction: ";
44                 cin >> key;
45                 if (key=='P') printSeat(movie1);
46                 if (key=='C') clearSeat(movie1);
47                 if (key=='B') {
48                         char ch;
49                         int sn;
50                         cout << "Enter the row and seat number: ";
51                         cin >> ch >> sn;
52                         buySeat(movie1,ch,sn);      }
53         } while (key != 'E');
54         return 0;
55 }
```

Buy if the seat is available, warn if it is not available

Use sentinel values to make instructions