Array

Inst. Nguyễn Minh Huy

Contents

- Array 1-D.
- Array 2-D.

Array 1-D

- Consider a program:
  - Enter 5 integers, then print them out.
    - Declare 5 ints a1, a2, a3, a4, a5.
  - Enter 50 integers, then print them out.
    - Declare 50 ints!

  ➔ How to declare many variables at once?

  ➔ Use array.

Array 1-D

- Basic concepts:
  - A sequence of contiguous variables of the same type.
  - Each variable in array called array element.
  - Declaration:

    <data type> **<array name>[** <array length> **]**;

    <array length>: number of elements, must be constant.

```
int    m1[ 10 ];          // Array of 10 integers.
float  m2[ 50 ];          // Array of 50 floats.

int    N;
float  m3[ N ];           // Wrong.

const int K = 100;
float  m4[ K ];           // Array of 100 floats.
```

Array 1-D

- Basic concepts:
  - Uninitialized array has random element values.

int **m[5]**;	**m**	?	?	?	?	?	

  - Initialization:

<data type> **<array name>[**<array length>**]** = { **<value 1>**, **<value 2>**, … };

int **m1[5]** = { 1, 2, 3, 4, 5 };	**m1**	1	2	3	4	5	// Initialize all elements.
int **m2[5]** = { 1, 2 };	**m2**	1	2	0	0	0	// Initialize some elements. // remain elements = 0
int **m3[5]** = { 0 };	**m3**	0	0	0	0	0	// Initialize all elements = 0
int **m4[ ]** = { 1, 2, 3, 4, 5 };	**m3**	1	2	3	4	5	// Initialize with auto length

Array 1-D

- Basic concepts:
  - Accessing element:

    <array name> **[<index>]**

    <index>: integer in range **[0.. <array length> - 1]**.

```
int a[ 10 ] = { 0 };
```

	0	1	2	3	4	5	6	7	8	9
a	0	0	0	0	0	0	0	0	0	0

```
a[ 0 ] = 5;
a[ 1 ] = 6;
a[ 2 ] = a[ 0 ] + a[ 1 ];

a[ -1 ] = 7;                // Wrong.
a[ 10 ] = 8;                // Wrong.
```

Array 1-D

- Basic concepts:
  - Passed as argument:
    - Method 1: void foo(**int a[100], int size**);

      ➔ Static array, accept only array of 100 elements.
    - Method 2: void foo(**int a[ ], int size**);

      ➔ Dynamic array, accept array of any length.
    - Array elements can be CHANGED.

```
void foo( int a[ ], int size )
{
        a[ 2 ] = 9;
        a[ 5 ] = 8;
}
```

```
int main()
{
        int  a[ 100 ] = { 0 };
        int  b[ 200 ] = { 0 };
        foo( a, 50 );
        foo( b, 70 );
        // a[2], a[5] changed.
        // b[2], b[5] changed.
}
```

Array 1-D

- Operations:
  - Iterate each element:
    - C **loop** + **counter**:
      - Iterate N elements.
      - Access element by index.
    - C++11 **ranged for** (foreach):
      - Iterate ALL elements.
      - Access element by reference.
    - C++20 **ranged for** with span:
      - Iterate N elements.
      - Access element by reference.
      - Library <ranges>.

```
// C loop + counter: N elements.
for ( int i = 0; i < N; i++ )
{
    <Access element A[ i ] by index>
}

// C++11 ranged for: ALL elements.
for ( int  &e: A )
{
    <Access element e by reference>
}

// C++20 ranged for with span.
for ( int  &e: std::span { A, N } )
{
    <Access element e by reference>
}
```

Array 1-D

- Operations:

```
void inputArray( int a[ ], int &n )
{
        printf(“Number of elements = “);
        scanf(“%d”, &n);

        for (int i = 0; i < n; i++)
        {
              printf(“Element %d = “, i);
              scanf(“%d”, &a[ i ]);
        }
}
void printArray( int a[ ], int n )
{
        for ( int i = 0; i < n; i++ )
              printf(“%d “, a[ i ]);
}
```

```
#define MAX    1000

int main()
{
        int  m[ MAX ], size;

        inputArray(m, size);
        printArray(m, size);
}
```

Array 1-D

- Operations:

```
// Sum of elements in a length n.
long sumArray( int a[ ], int n )
{
        long  sum = 0;

        for ( int i = 0; i < n; i++ )
              sum += a[ i ];

        return sum;
}
```

```
#define MAX    100

int main()
{
        int  m[ MAX ], size;

        inputArray(m, size);
        long  tong = sumArray(m, size);
}
```

Array 1-D

- Operations:
  - Copy array:

```
int  a [ MAX ] = { 1, 2, 3, 4, 5 };
int  b [ MAX ];
b = a;                        // Wrong, cannot copy by assignment!
copyArray( a, n, b );         // Write function to copy a length n to b.
```

  - Insert element:

```
int  a [ MAX ] = { 1, 2, 4, 5 };
int  x = 3;
int  pos = 2;                 // Array a length n, insert x at index pos.
insertArray( a, n, x, pos );  // a = { 1, 2, 3, 4, 5 }
```

  - Delete element:

```
int  a [ MAX ] = { 1, 2, 3, 4, 5 };
int  pos = 2;                 // Array a length n, delete element at index pos.
deleteArray( a, n, pos );     // a = { 1, 2, 4, 5 }
```

Array 1-D

- Array of struct:

```
struct Student  m1[ 10 ];               // Array of 10 struct Student.

struct Student  m2[ 10 ] = {            // Initialize some array elements.
    { “24127001“, “minh”, 8.5, 9.0 },
    { .name = “trang”, .literature = 9.5 }
};

struct Student  m2[  ] = {              // Auto array length.
    { “24127001”, “minh”, 8.5, 9.0 },
    { .name = “trang”, .literature = 9.5 }
};
```

Contents

- Array 1-D.
- **Array 2-D.**

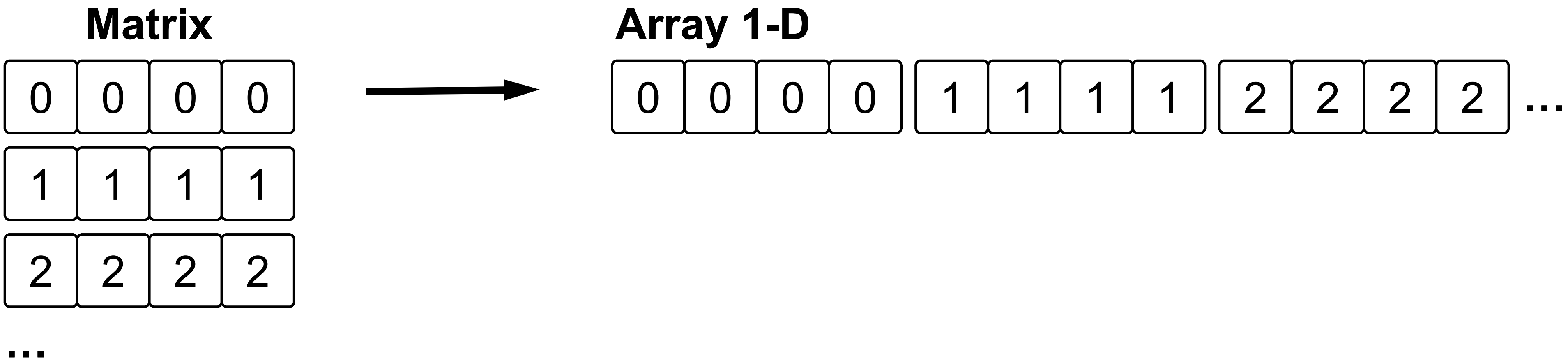
Array 2-D

- Consider the program:
  - Enter a matrix of 5 x 10 integers, then print it out.
    - Declare 5 arrays: int a1[10], a2[10], a3[10], a4[10], a5[10].
  - Enter a matrix of 50 x 10 integers, then print it out.
    - Declare 50 arrays!

  ➔ How to declare a matrix of M x N?

Array 2-D

- Method 1:
  - Use array 1-D!
  - Matrix M x N ~ array 1-D of M x N elements.
  - Matrix [row i, column j] ~ array [i * N + j].



Matrix

0	0	0	0
1	1	1	1
2	2	2	2

...

Array 1-D

0	0	0	0	1	1	1	1	2	2	2	2	...

Array 2-D

- Method 2:
  - Use array 2-D.
  - Declaration:

    <data type> **<array name>[**<rows>**] [**<columns>**]**;

    <rows>, <columns> must be constants.

    ```
    int    m1[ 5 ][ 10 ];  // Matrix 5 x 10 of integers.
    int    m2[ M ][ N ];   // Wrong.
    ```

  - Accessing element:

    <array name> **[<row index>] [<column index>]**

    <row index>: integer in range **[0.. <rows> - 1]**.

    <column index>: integer in range **[0.. <columns> - 1]**.

    ```
    m1[ 0 ][ 2 ] = 5;
    m1[ 1 ][ 3 ] = 6;
    m1[ -1 ][ 10 ] = 7;    // Wrong.
    ```

Array 2-D

- Initialization:

```
<data type> <array name>[<rows>] [<columns>] =
{
          <Initialize row 0>,
          <Initialize row 1>,
          …
};
```

```
 // Initialize all elements.
int  m1[ 3 ][ 5 ] =
{
      { 1, 1, 1, 1, 1 },
      { 1, 2, 3, 4, 5 },
      { 5, 4, 3, 2, 1 }
};
```

```
 // Initialize some elements.
int  m1[ 3][ 5 ] =
{
      { 1, 1 },
      { 1, 2, 3 },
      { 0 }
};
```

```
 // Initialize with auto rows.
int  m1[  ][ 5 ] =
{
      { 1, 1 },
      { 1, 2, 3 },
      { 0 }
};
```

Array 2-D

- Operations:
  - Step 1: Iterate through array.
    - Use **2 nested loops**.
    - Outer loop: iterate through rows.
    - Inner loop: iterate through columns in each row.
  - Step 2: Access element by indexing.

```
// Iterate matrix A of M rows N columns.
for ( int i = 0; i < M; i++ )
        for ( int j = 0; j < N; j++ )
        {
                <Process element A[ i ][ j ]>;
        }
```

Array 2-D

- Operations:

```
// Input matrix A of M rows N columns.
void inputMatrix( int A[ ][ MAX ], int &M, int &N )
{
    printf(“Enter rows, columns = “);
    scanf(“%d %d”, &M, &N);

    for (int i = 0; i < M; i++)
       for (int j = 0; j < N; j++)
       {
           printf(“Element %d, %d = “, i, j);
           scanf(“%d”, &A[ i ][ j ]);
       }
}
```

```
#define MAX    100

int main()
{
        int  a[ MAX ][ MAX ];
        int  M, N;

        inputMatrix(a, M, N);
}
```

Summary

- Array 1-D:
  - Sequence of contiguous elements of same type.
  - Element accessed by indexing.
  - Operations: use loop to iterate through elements.
- Array 2-D:
  - Method 1: array 1-D of rows x columns elements.
  - Method 2: array 2-D.
  - Matrix of elements of same type.
  - Operations: use nested loop to iterate.

Practice

- Practice 5.1:

  Write C/C++ program (organize in functions and multiple-file project):

  - Enter an array of integers.
  - Count:

    a) Negative numbers in the array.

    b) Prime numbers in the array.

  Input format:

  Enter N = 3

  Element 0 = 2

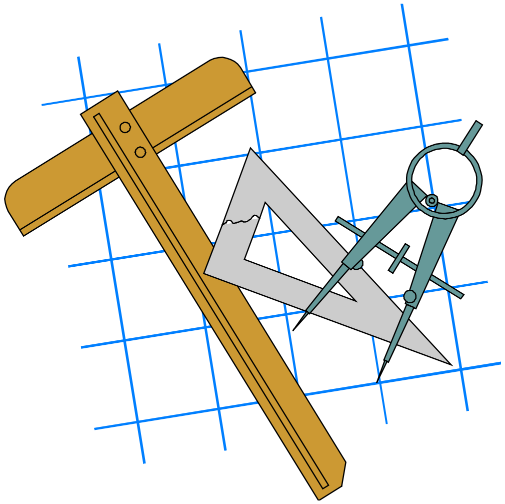
  Element 1 = 3

  Element 2 = -6

  Output format:

  Negative numbers: 1.

  Prime numbers: 2.



Practice

- ## Practice 5.2:

  Write C/C++ program to check array:

  (organize in functions and multiple-file project)

  - Enter a array of integers.

  - Check if:

    a) Array is in ascending order.

    b) Array is symmetric.

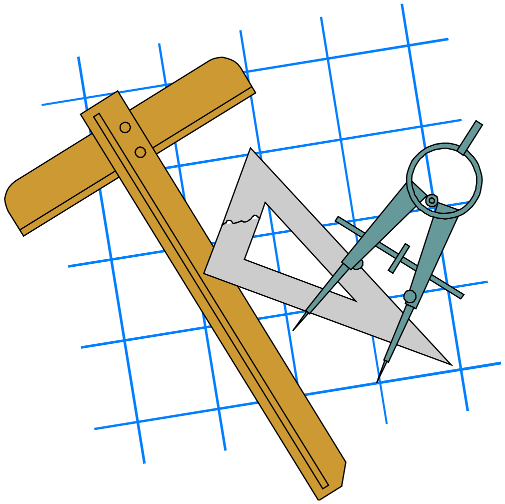
    c) Array is an arithmetic progression.

  Output format:

  Array <is/is not> ascending.

  Array <is/is not> symmetric.

  Array <is/is not> a arithmetic progression.



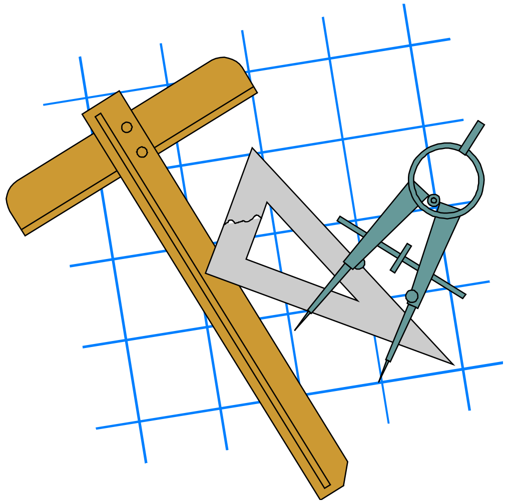
Practice

- Practice 5.3:

  Write C/C++ program to operate on students:

  (organize in functions and multiple-file project):

  - Declare struct to represent a student (stated in the lesson).
  - Enter a list of N students.
  - Print a list of excellent students (GPA >= 8.5) in descending order of GPA.



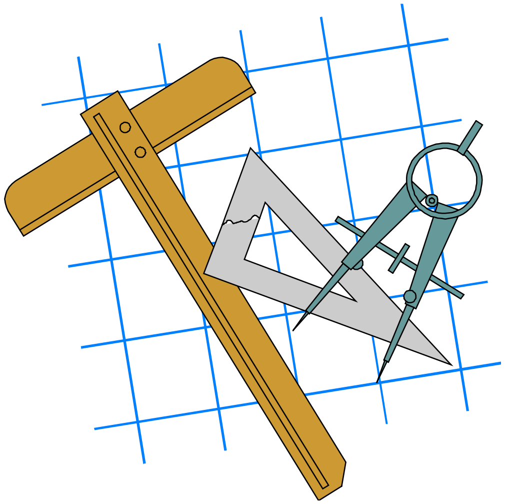
Practice

- ## Practice 5.4:

Write C/C++ program to operate on date:

(organize in functions and multiple-file project):

- Declare struct to represent date (day, month, year).
- Enter a list of date.
- Print the list from the latest date to the oldest one.



Practice

- ## Practice 5.5:

  Write C/C++ program as follow:

  (organize in functions and multiple-file project)

  - Enter square matrix N x N of integers.
  - Print to screen:

    a) Sum of elements on each diagonal.

    b) Row index having max sum of elements.

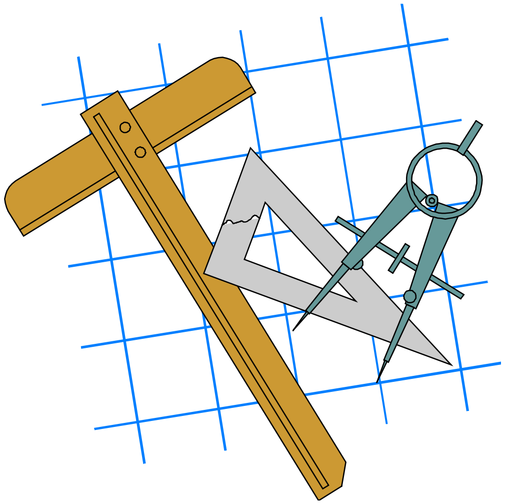
    c) It is a magic square or not.

Output format:

Main diagonal = <sum of elements>.

Anti-diagonal = <sum of elements>.

It <is/is not> a magic square.



Practice

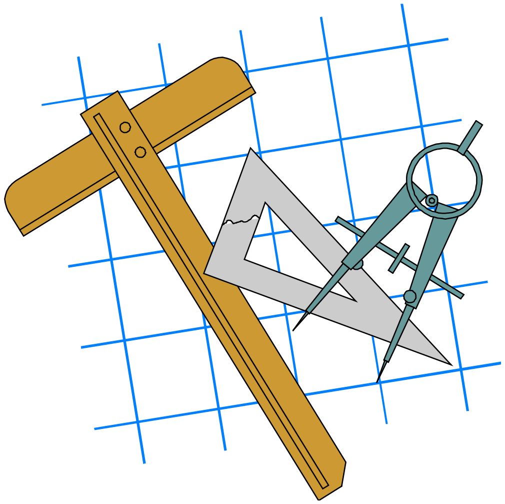
- ## Practice 5.6:

  Write C/C++ program to manipulate matrix:

  (organize in functions and multiple-file project)

  - Enter a matrix M x N of integers.

  - Print to screen all elements satisfying its value equals to sum of the remaining elements on its row and column.



Practice

- Practice 5.7:

  Write C/C++ program to rotate matrix:

  (organize in functions and multiple-file project)

  - Enter a matrix M x N of integers.
  - Rotate left matrix 90 degree and print result.
  - Rotate right matrix 90 degree and print result.