ensia
The National School of
Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

A
R
R
A
Y
S
0
1
2
3
4
5
Indices

Data Structure & Algorithms 1

CHAPTER 4
STATIC DATA STRUCTURE (PART1): 1D ARRAY

Sep – Dec 2023

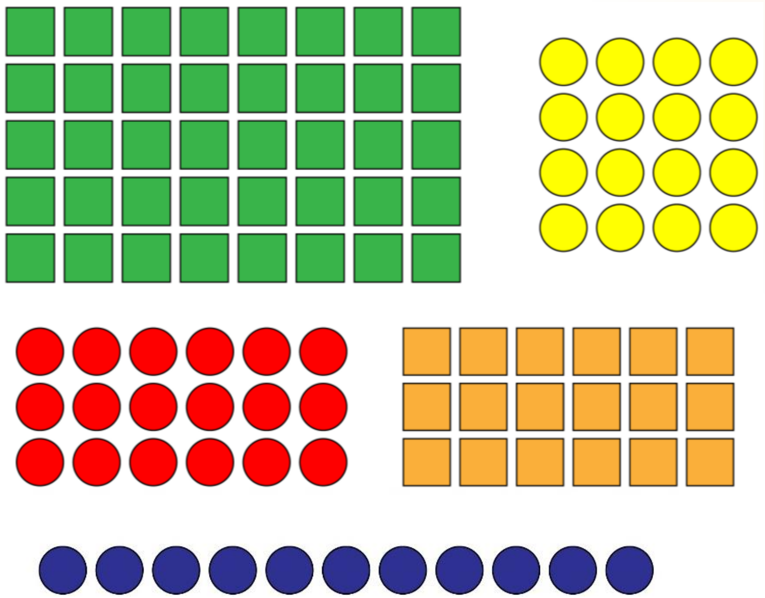
Outline

- Introduction
  - Definition
  - Example
- Fundamental Modules for 1D Array Operations
  - Read, Write
  - Applications
    - Searching
  - Other modules
    - Random insertion
- 1D Array in C++
  - Definition and Declaration
  - Examples

WHAT IS ARRAY?

Group Of

Objects



WHAT IS ARRAY?

A Data Structure containing a Number of data values, all of which of same Type

Format for organizing and storing data

This array consist of 9 data values

2	4	-5	3	0	25	6	2	1

All elements in the array are of the same data-type (int/char/…)

Introduction to 1D Array

Definition

- **1D ARRAY:** is an object composed of multiple elements of the **same type**, and each element is identified by an **index**.

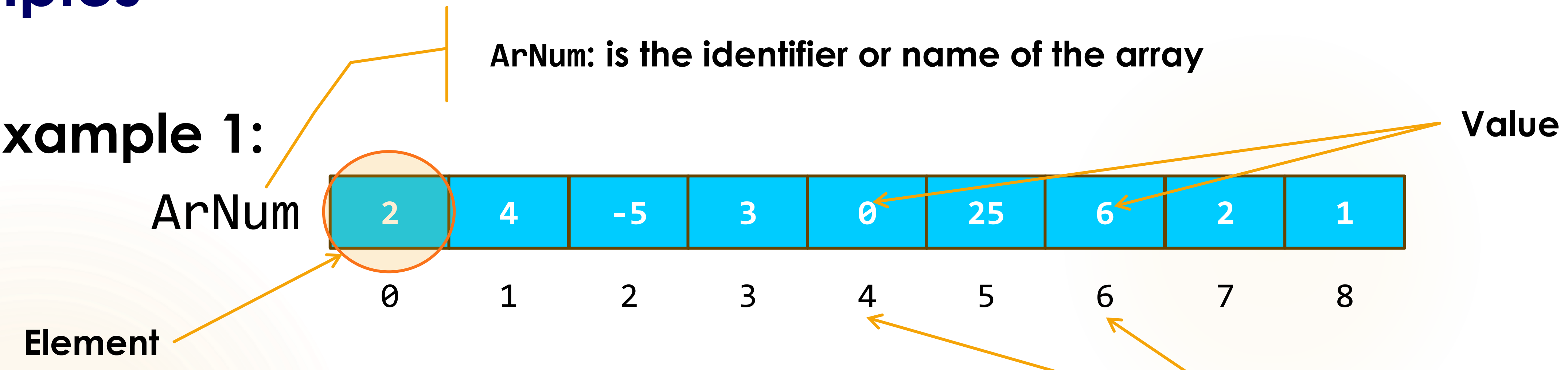
  The **1D-ARRAY** is a **STATIC** & **SEQUENCIAL** data structure

- Accessing an element of the array is done by specifying the name of the array followed by [*the index value within brackets*]

Introduction to 1D Array

Examples

- **Example 1:**



ArNum: is the identifier or name of the array

ArNum	2	4	-5	3	0	25	6	2	1
	0	1	2	3	4	5	6	7	8

This array consists of 9 elements of type Integer

- `ArNum[4]:` Represents the 5th element of `ArNum`, which is the value `0`

Introduction to 1D Array

Examples

- **Example 2:** Let's consider an array named COEF containing the coefficients of 7 courses

COEF	2	4	5	3	1	2	5
	0	1	2	3	4	5	6

We can write the following actions: `A = COEF[3]`

- **Example 3:**

```
C = T1[(test+k)MOD i]; T2[7] = E; T[m] = F;
T4[a*res DIV 2] = G
```

Introduction to 1D Array

- Through these examples, we observe that:
  - an element of the array is treated as a variable, and
  - the index value can be a constant, a variable, or an expression.
- The number of elements in the array determines its **size**.
- The number of indices used to identify a specific element is called the **dimension** of the array.
- The type of the index is a scalar type (and often an interval)

Declaration of 1D Array

To define an array, you need to:

- Know the number of its elements, i.e., its size.
- Determine the type of each element.
- Provide the name of the array.

The declaration of an array is done by using the:

- element type followed by the array name **[**array_size**]** within **brackets**.

Declaration of 1D Array

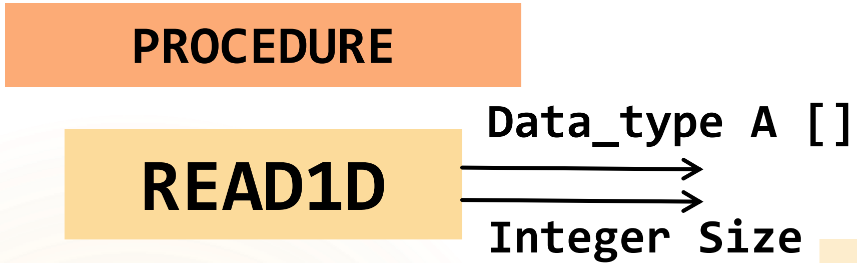
- **FORMAT:**

```
Element_type  ArrayName [array_size]
```

- **Example:**

```
Char message [100] // to store a message with a max of 100 char
Boolean flag [8] // to track the status of 8 different conditions
Integer temperature [24] // to store temperature over 24h hours
```

Basic Modules for 1D Array



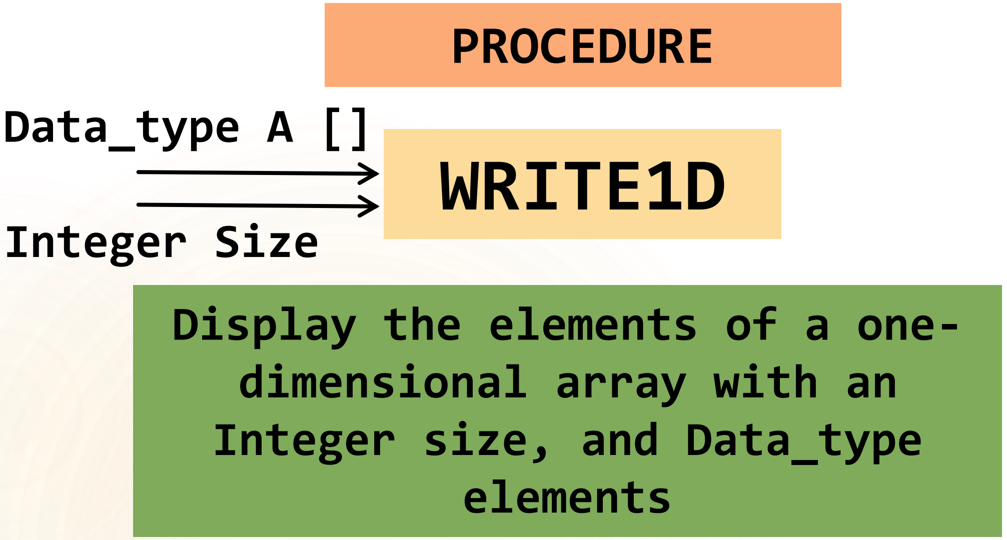
It read the size (Size) and elements (Integer) of a one-dimensional array ([])

ANALYSIS

- The size (Size) of the array is read.
- We vary i = 0, 1, 2, .., Size-1, and in each iteration:
  - We read the element A[i] of the array.

```
Procedure READ1D (Var Data_type A[], Var Integer Size)
Variable Integer i
BEGIN
    WRITE (‘Enter the size of the array’)
    READ (Size)
    FOR i FROM 0 TO Size-1 DO
        WRITE (‘A[‘, i, ‘] = ’)
        READ (A[i])
    END FOR
END
```

Basic Modules for 1D Array



Display the elements of a one-dimensional array with an Integer size, and Data_type elements

ANALYSIS

- We vary i = 0, 1, 2, .., Size-1, and in each iteration:
    - We write the element A[i] of the array.

```
Procedure WRITE1D (Data_type A[], Integer
Size)
Variable Integer i
BEGIN
    FOR i FROM 0 TO Size-1 DO
        WRITE (A[i], ‘ |’)
    END FOR
END
```

Basic Modules for 1D Array

Application 1: Searching for an element in a 1D array

Provide the solution that allows searching for a given value **V** in an array of a maximum of 100 integers

Modular Decomposition:

- We need a module SearchV
- Basic modules:
    - READ1D and WRITE1D

Module SearchV

//Allows to determine if V is in the array or not

Analysis:

- Initialization of a variable Found to false;
- Initialization of a variable i;
- While (i < Zise) AND Found = False)
    - Compare an element A[i] with the value V
        - If A[i] equals V, set Found to True
- Assignment: **SearchV** = Found

Basic Modules for 1D Array

Application 1: Searching for an element in a 1D array

```
Boolean Function SearchV (Data_type A[], Integer Size, Data_type V)
Variable Integer i
         Boolean Found
BEGIN
    Found = False
    i = 0
    WHILE (i < Size) AND (Found == False) DO
         IF A[i] == V THEN
             Found = True
         END IF
         i = i + 1
    END WHILE
    SeachV = Found
END
```

Basic Modules for 1D Array

Application 1: Searching for an element in a 1D array

Main Program

Analysis:

- We call the procedure READ1D to read the size of the array and its elements.
- We read the searched value (**V**).
- We call the function **SearchV** (`Result = SearchV(A, Size, V)`).
- We display a message (element found or not found).

Basic Modules for 1D Array

Application 1: Searching for an element in a 1D array

```
Algorithm Application_1
Constant MAX = 100
Variable Integer A[MAX], V, Size // Example: Data_type equals to Integer
         Boolean Result
Procedure READ1D (Var Integer A[], Var Integer Size)
….
Boolean Function SearchV (Integer A[], Integer Size, Integer V)
….
BEGIN
   READ1D(A, Size)
   WRITE(‘The searched value? : ’)
   READ(V)
   Result = SearchV(A, Size, V)
   IF Result == True Then
      WRITE (‘ Exist in the Array’)
   ELSE
      WRITE (‘ Does not exist in the Array’)
   END IF
END
```

Basic Modules for 1D Array

Application 2: Count the number for an element in a 1D array

Find the number of elements equal to a given value V in an array of a maximum of 100 integer numerical elements

Modular Decomposition:

- We need a module **NbeV** to calculate the number of elements in the array that are equal to V, and
- basic modules:
    - READ1D and WRITE1D.

Module NbeV

//Calculate the number of elements in A equal to V

Analysis:

- Count = 0 (the frequency of appearance of V in A);
- We vary i = 0, 1, 2 ,.. Size-1, and for each iteration:
    - Compare an element A[i] with the value V
        - If A[i] equals V, we increment Count by one
- Assignment: **NbeV** = Count

Basic Modules for 1D Array

Application 2: Count the number of an element in a 1D array

```
Integer Function NbeV (Data_type A[], Integer Size, Data_type V)
Variable Integer i, Count
BEGIN
   Count = 0
   FOR i FROM 0 TO Size-1 DO
      IF A[i] == V THEN
         Count = Count + 1
      END IF
   END FOR
   NbeV = Count
END
```

Basic Modules for 1D Array

Application 2: Count the number of an element in a 1D array

Main Program

Analysis:

- We call the procedure READ1D to read the size of the array and its elements.
- We read the searched value (**V**).
- We call the function **NbeV** (`Result = NbeV(A, Size, V)`).
- We display the number of elements (`Result`) equal to V.

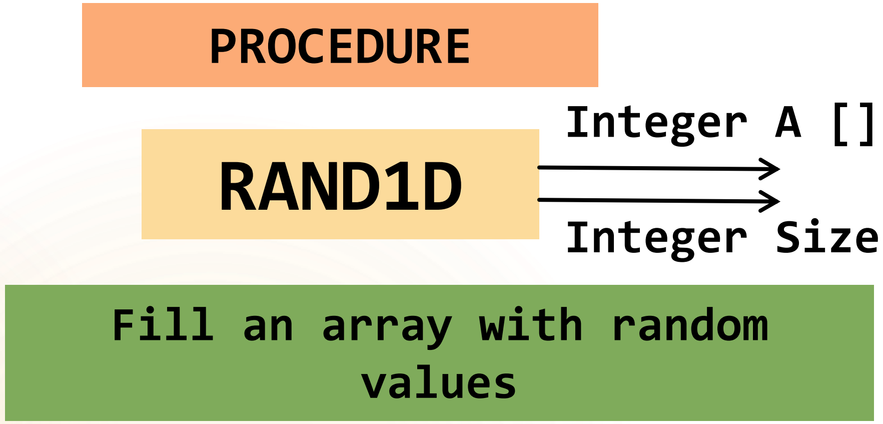
Basic Modules for 1D Array

Application 2: Count the number of an element in a 1D array

```
Algorithm Application_2
Constant MAX = 100
Variable Integer A[MAX], V, Size, Result // Example: Data_type equals to Integer

Procedure READ1D (Var Integer A[], Var Integer Size)
….
Integer Function NbeV (Integer A[], Integer Size, Integer V)
….
BEGIN
   READ1D(A, Size)
   WRITE('The searched value? : ')
   READ(V)
   Result = NbeV(A, Size, V)
   WRITE ('There is ', Result, 'elements of ', V,)
END
```

Basic Modules for 1D Array

Other basic modules: RAND1D (Random insertion)



```
Procedure RAND1D (Var Integer A[], Var Integer Size)
Variable Integer i, minI, maxI
BEGIN
    WRITE (‘Enter the size of the array’)
    READ (Size)
    WRITE (‘[minI, maxI] random values’)
    READ (minI, maxI)
    FOR i FROM 0 TO Size-1 DO
        A[i] = minI + Random((maxI-minI))
    END FOR
END
```

Generate random numbers between 0 and ((maxI-minI))

ANALYSIS

- Generate random numbers and insert them int array A

Basic Modules for 1D Array

Other Applications

- Complete the previous module with the search and display of minimum (minArray1D) and maximum (maxArray1D) of an array.
- Build a library **LArray1D** that encompasses the procedures READ1D, WRITE1D, RAND1D, and the functions minArray1D, maxArray1D, NbeV.
- Test this library using a menu MArray1D.

1D Array in C++

Definition

In C++ an Array is considered as:

- Structures of related data items
- Static entity (same size throughout program)

- Consecutive group of memory locations
- Same name and type (int, char, etc.)

1D Array in C++

Declaration

To refer to an element

- Specify array name and position number (index)
- Format: `arrayName[ index ]`
- First element at position `0`

N-element array c

- `c[ 0 ], c[ 1 ] … c[ n - 1 ]`
- Nth element at position `N-1`

1D Array in C++

Declaration

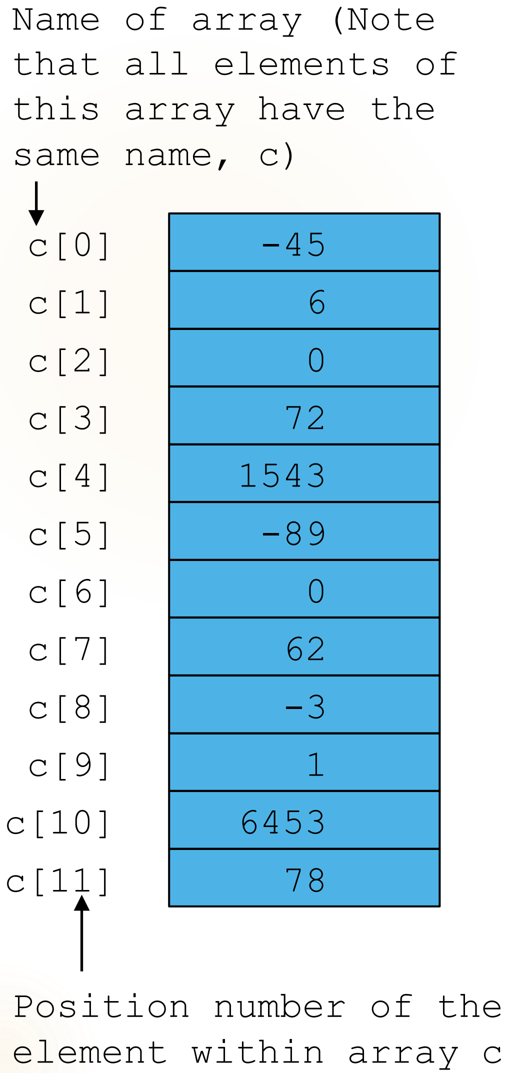
Array elements like other variables

- Assignment, printing for an integer array

```
cc[0] =  3;
cout << c[0];
```

- Can perform operations inside subscript

`c[5 - 2]` same as `c[3]`



Name of array (Note that all elements of this array have the same name, c)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

1D Array in C++

Declaration

When declaring arrays, specify:

- Name
- Type of array
  - Any data type
- Number of elements
- *`type arrayName[arraySize];`*

```
int c[10];  // array of 10 integers
float d[3284]; // array of 3284 floats
```

Declaring multiple arrays of same type

- Use comma separated list, like regular variables

```
int b[ 100 ], x[ 27 ];
```

1D Array in C++

Example using array

Initializing arrays

- For loop
  - Set each element
- Initializer list
  - Specify each element when array declared

```
int n[5] = {1, 2, 3, 4, 5};
```

  - If not enough initializers, rightmost elements 0
  - If too many syntax error
- To set every element to same value

```
int n[5] = {0};
```

- If array size omitted, initializers determine size

```
int n[] = {1, 2, 3, 4, 5};
```

  - 5 initializers, therefore 5 element array

1D Array in C++

Example using array

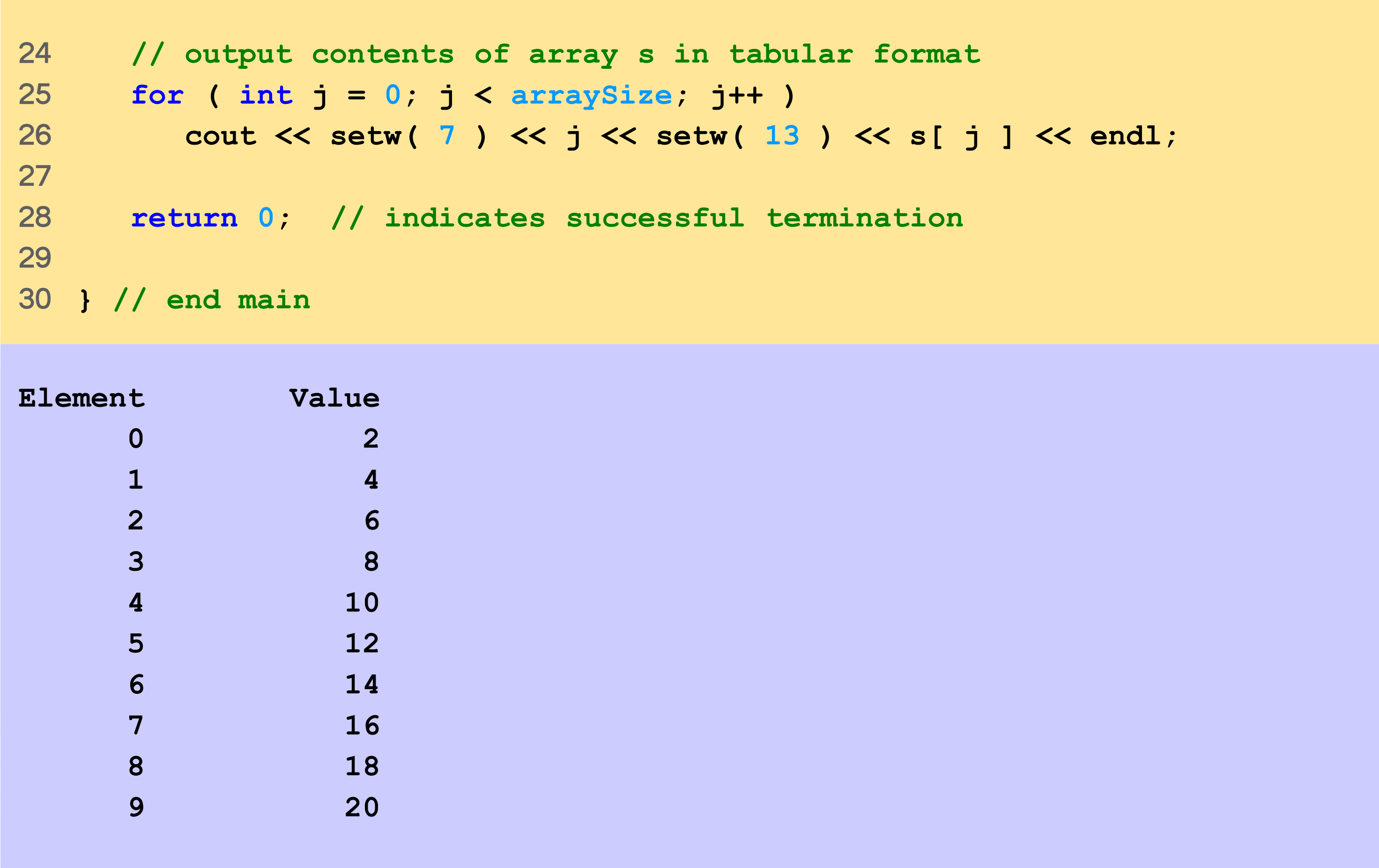
```cpp
// Fig. 4.5: fig04_05.cpp
// Initialize array s to the even integers from 2 to 20.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   // constant variable can be used to
   const int arraySize = 10;

   int s[ arraySize ]; // array s has 10 el

   for ( int i = 0; i < arraySize; i++ )  //
      s[ i ] = 2 + 2 * i;

   cout << "Element" << setw( 13 ) << "Value

```

Note use of **`const`** keyword. Only **`const`** variables can specify array sizes.

The program becomes more scalable when we set the array size using a **`const`** variable. We can change **`arraySize`**, and all the loops will still work (otherwise, we'd have to update every loop in the program).

1D Array in C++

Example using array



```
   // output contents of array s in tabular format
   for ( int j = 0; j < arraySize; j++ )
      cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

   return 0;  // indicates successful termination

} // end main
```

```
Element        Value
      0            2
      1            4
      2            6
      3            8
      4           10
      5           12
      6           14
      7           16
      8           18
      9           20
```