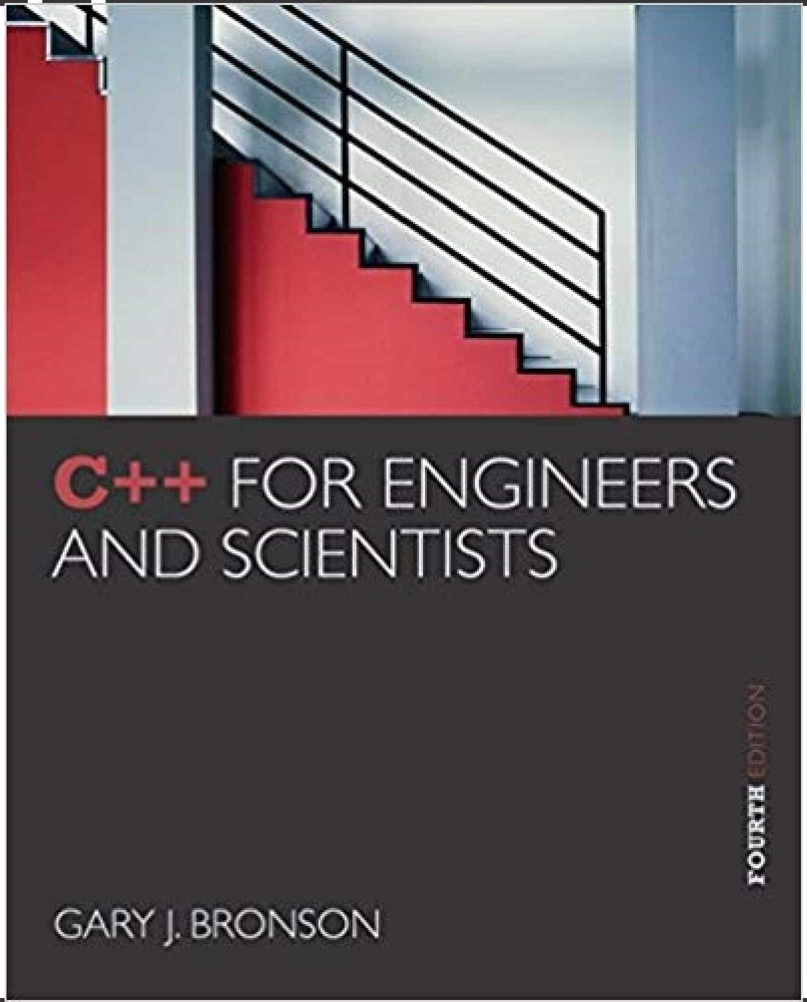
ELEG 1043
Computer Applications in Engineering



Chapter 7: Arrays

C++ FOR ENGINEERS AND SCIENTISTS

Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Objectives

In this chapter, you will learn about:

- One-dimensional arrays
- Array initialization
- Declaring and processing two-dimensional arrays
- Arrays as arguments
- Statistical analysis

Objectives (continued)

- The Standard Template Library (STL)
- Searching and sorting
- Common programming errors

One-Dimensional Arrays

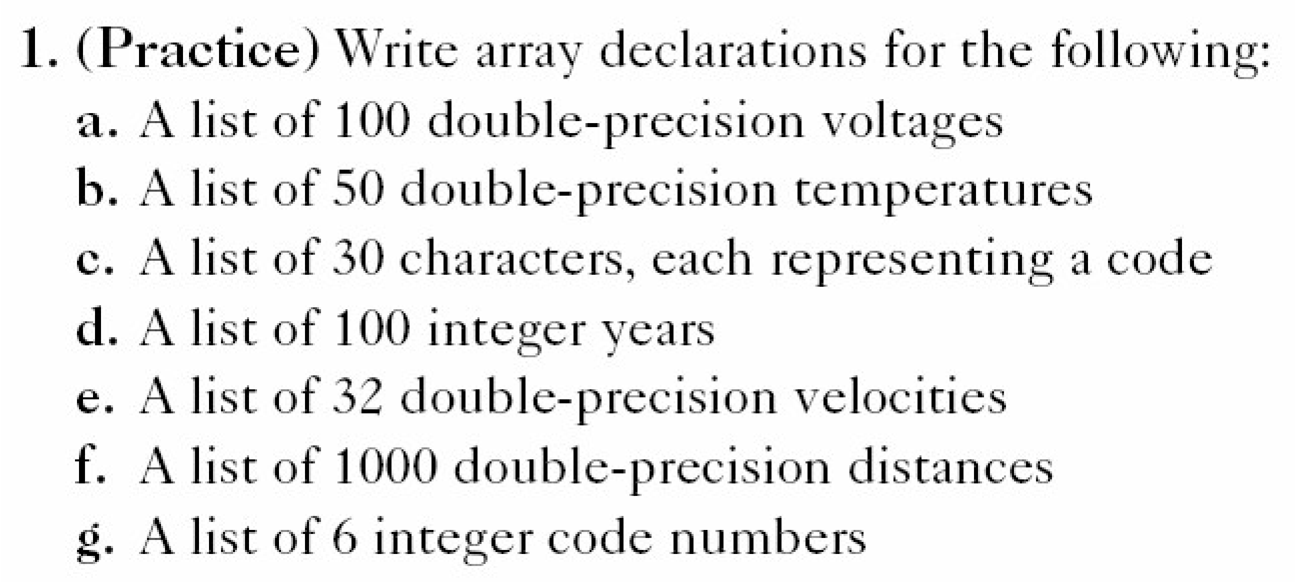
- **One-dimensional array:** A list of related values with the same data type, stored using a single group name (called the **array name**)
  - Syntax:

    ```
    dataType arrayName[number-of-items]
    ```

- By convention, the number of items is first declared as a constant, and the constant is used in the array declaration

Homework Assignment 4

- EXERCISES 7.1 in the textbook



1. (Practice) Write array declarations for the following:
   a. A list of 100 double-precision voltages
   b. A list of 50 double-precision temperatures
   c. A list of 30 characters, each representing a code
   d. A list of 100 integer years
   e. A list of 32 double-precision velocities
   f. A list of 1000 double-precision distances
   g. A list of 6 integer code numbers

One-Dimensional Arrays (continued)

- **Element**: An item in the array
  - Array storage of elements is contiguous
- **Index** (or **subscript**) of an element: The position of the element within the array
  - Indexes are zero-relative
- To reference an element, use the array name and the index of the element

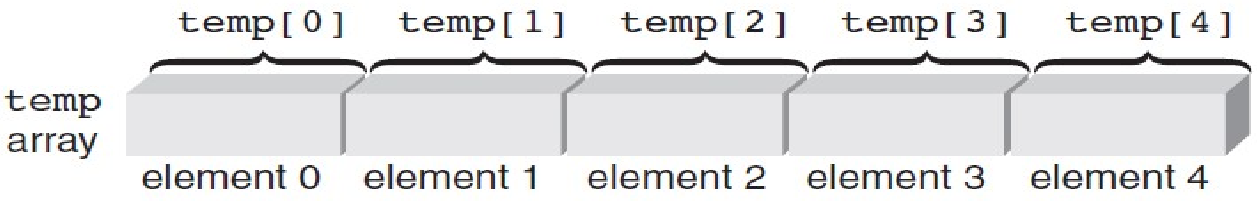


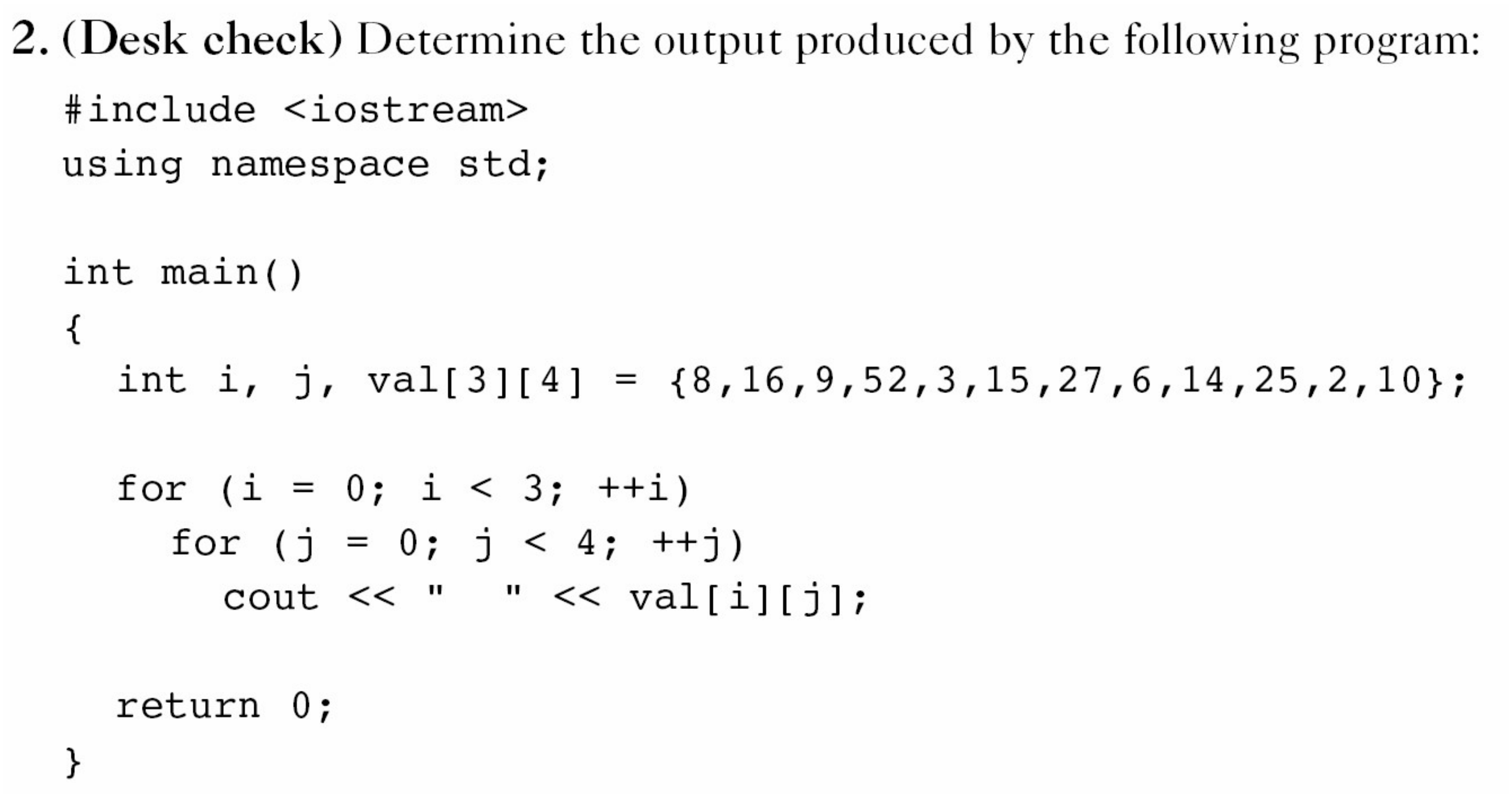
Figure 7.2 Identifying array elements

One-Dimensional Arrays (continued)

- All of the elements of an array can be processed by using a loop
- The loop counter is used as the array index to specify the element
- Example:

```
 int sum = 0;
    int temp[5] = {1,2,3,4,5};
 for (int i=0; i<5; i++)
     sum = sum + temp[i];
```

Homework Assignment 4



2. (Desk check) Determine the output produced by the following program:

```
#include <iostream>
using namespace std;

int main()
{
  int i, j, val[3][4] = {8,16,9,52,3,15,27,6,14,25,2,10};

  for (i = 0; i < 3; ++i)
    for (j = 0; j < 4; ++j)
      cout << "  " << val[i][j];

  return 0;
}
```

Homework Assignment 4

```
int a[20] =
   {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
for (int k = 1; k < 5; k = k + 3)
      cout << a[k] << " ";
```

Input and Output of Array Values

- Array elements can be assigned values interactively using a **`cin`** stream object
- Out of range array indexes are not checked at compile-time
  - May produce run-time errors
  - May overwrite a value in the referenced memory location and cause other errors
- Array elements can be displayed using the **`cout`** stream object

Declaring and Processing Two-Dimensional Arrays

- **Two-dimensional array:** Has both rows and columns
  - Also called a **table**
- Both dimensions must be specified in the array declaration
  - Row is specified first, then column
- Both dimensions must be specified when referencing an array element

Declaring and Processing Two-Dimensional Arrays (cont'd)

- Two-dimensional arrays can be initialized in the declaration by listing values within braces, separated by commas
- Braces can be used to distinguish rows, but are not required
- Nested `for` loops are used to process two-dimensional arrays
  - Outer loop controls the rows
  - Inner loop controls the columns

Arrays as Arguments

- An individual array element can be passed as an argument just like any individual variable
- The called function receives a copy of the array element’s value
- Passing an entire array to a function causes the function to receive a reference to the array, not a copy of its element values
- The function must be declared with an array as the argument
- Single element of array is obtained by adding an offset to the array’s starting location

Exercise 1

- Write a program to input 10 positive integer numbers in an array named **Mini** and determine and display the m**inNum** value entered, where the numbers are received from keyboard

Answer

```
#include <iostream>
using namespace std;
int main(){
    int Mini [10] = {0};
     int num = -1;
     cin>>num;
    Mini [0] = num;
    int minNum = num; \\ assign a large number
    for(int i = 1; i < 10; i++)
    {  Mini [i] = num;
       if(minNum > num)
       {minNum = num;}
    }
    cout<<minNum<<endl;
    return 0;}
```

Exercise 2

- Write a program to build a function named **multiply** to input the following integer numbers in an array named grades: 12.3, 16.4, and 30.6. As each number is input, multiply the numbers to a variable **mul** and return the **mul** value.

Answer

```
#include <iostream>
using namespace std;
double multiply(double arr[], int length);
int main(){
    double arr[3] = {12.3, 16.4, 30.6};
    cout<< multiply(arr, 3);
    return 0;
}

 double multiply(double arr[], int length){
     double mul = 1.0;
     for(int i = 0; i < length; i++){
         mul = mul*arr[i];
     }
     return mul;
 }
```

Summary

- An array is a data structure that stores a list of values having the same data type
  - Array elements: stored in contiguous memory locations; referenced by array name/index position
  - Two-dimensional arrays have rows and columns
  - Arrays may be initialized when they are declared
  - Arrays may be passed to a function by passing the name of the array as the argument
    - Arrays passed as arguments are passed by reference
    - Individual array elements as arguments are passed by value (copy)

Chapter 10: Pointers

C++ FOR ENGINEERS AND SCIENTISTS

Objectives

- In this chapter you will learn about:
  - Addresses and pointers
  - Array names as pointers
  - Pointer arithmetic
  - Passing addresses
  - Common programming errors

Addresses and Pointers

- The address operator, `&`, accesses a variable’s address in memory
- The address operator placed in front of a variable’s name refers to the address of the variable

  `&num` means the address of `num`

Storing Addresses (continued)

- Example statements store addresses of the variable `m`, `list`, and `ch` in the variables `d`, `tabPoint`, and `chrPoint`

```
d = &m;
tabPoint = &list;
chrPoint = &ch;
```

- `d`, `tabPoint`, and `chrPoint` are called pointer variables or pointers

Storing Addresses (continued)

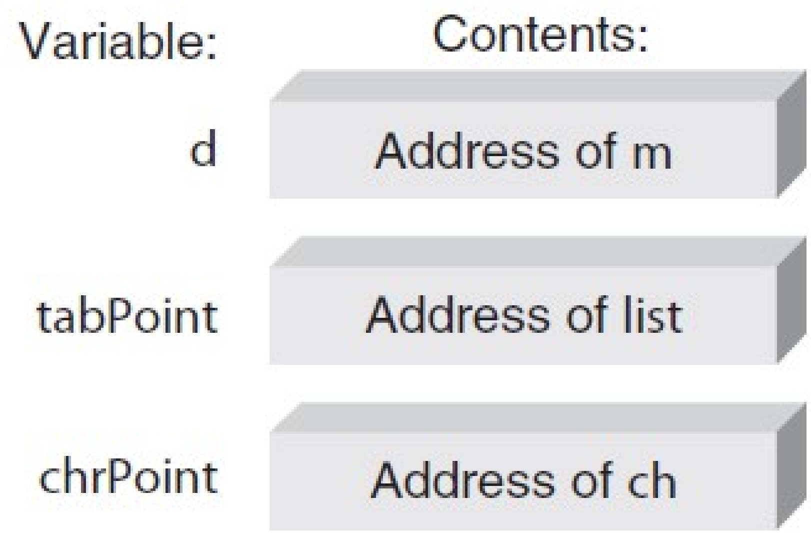


Figure 10.3 Storing more addresses

Declaring Pointers

- Like all variables, pointers must be declared before they can be used to store an address
- When declaring a pointer variable, C++ requires specifying the type of the variable that is pointed to
  - Example: `int *numAddr;`

Array Names as Pointers

- There is a direct and simple relationship between array names and pointers

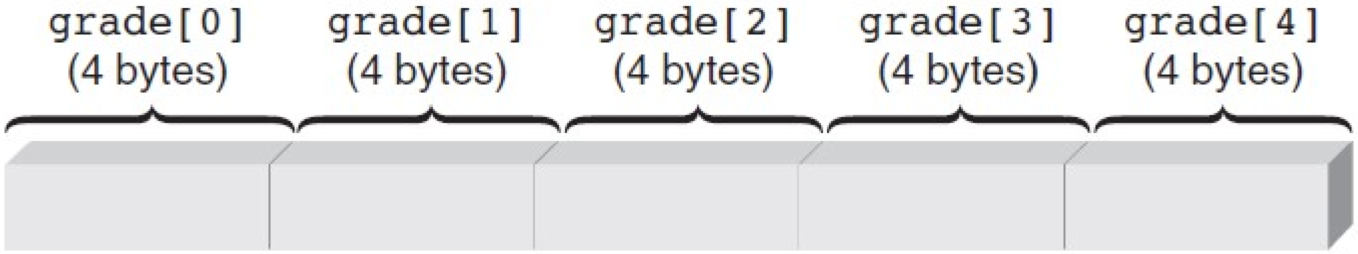
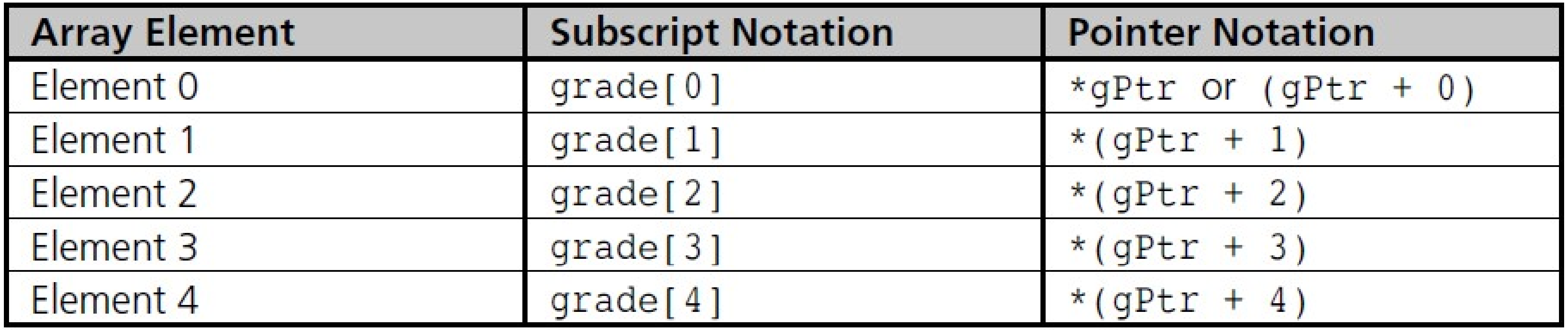


Figure 10.9 The `grade` array in storage

- Using subscripts, the fourth element in grade is referred to as grade[3], address calculated as:

```
&grade[3] = &grade[0] + (3 *
sizeof(int))
```

Array Names as Pointers (continued)



Array Element	Subscript Notation	Pointer Notation
Element 0	`grade[0]`	`*gPtr` or `(gPtr + 0)`
Element 1	`grade[1]`	`*(gPtr + 1)`
Element 2	`grade[2]`	`*(gPtr + 2)`
Element 3	`grade[3]`	`*(gPtr + 3)`
Element 4	`grade[4]`	`*(gPtr + 4)`

Table 10.1 Array Elements Can Be Referenced in Two Ways

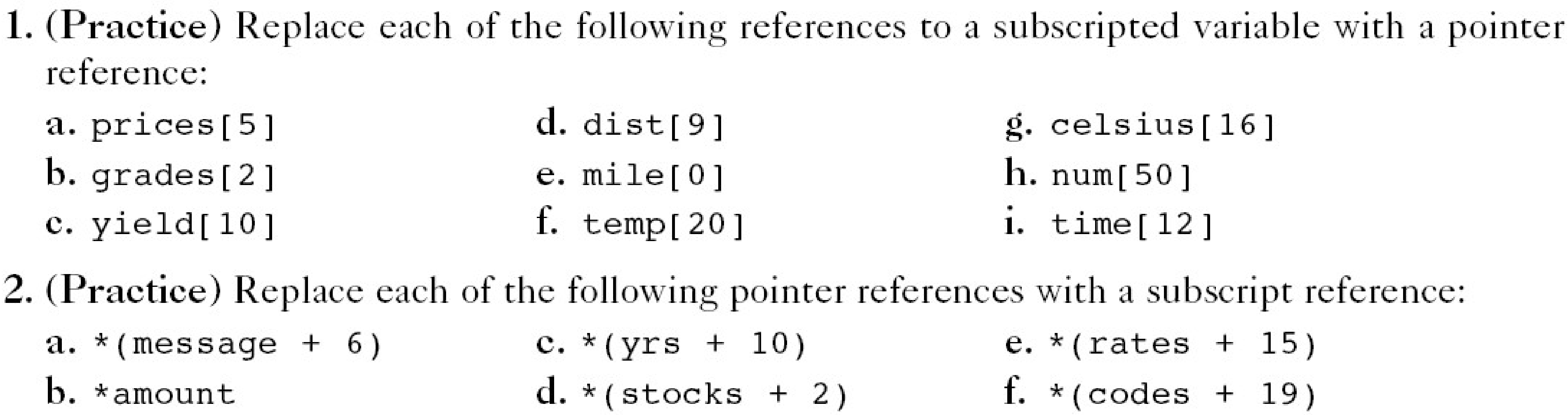
Exercise 1

- Replace each of the following references to a subscripted variable with a **pointer** reference
  - `year[10]`
  - `seconds[30]`
  - `students[0]`

Exercise 2

- Replace each of the following pointer references with a subscript (index) reference
  - `*(year + 2)`
  - `*(seconds + 20)`
  - `*(students)`

Homework Assignment 5



1. **(Practice)** Replace each of the following references to a subscripted variable with a pointer reference:

   a. `prices[5]`
   b. `grades[2]`
   c. `yield[10]`
   d. `dist[9]`
   e. `mile[0]`
   f. `temp[20]`
   g. `celsius[16]`
   h. `num[50]`
   i. `time[12]`

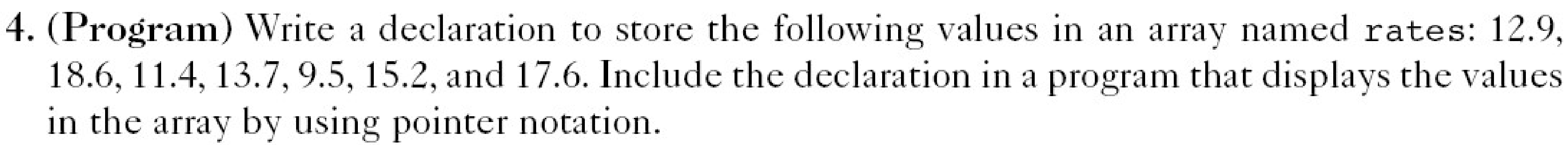
2. **(Practice)** Replace each of the following pointer references with a subscript reference:

   a. `*(message + 6)`
   b. `*amount`
   c. `*(yrs + 10)`
   d. `*(stocks + 2)`
   e. `*(rates + 15)`
   f. `*(codes + 19)`

Passing Arrays

- When an array is passed to a function, its address is the only item actually passed
  - "Address" means the address of the first location used to store the array
  - First location is always element zero of the array

Homework Assignment 5



4. **(Program)** Write a declaration to store the following values in an array named `rates`: 12.9, 18.6, 11.4, 13.7, 9.5, 15.2, and 17.6. Include the declaration in a program that displays the values in the array by using pointer notation.

Answer

```
#include <iostream>
using namespace std;
int main(){
    double arr[7] = {12.9,18.6,11.4,13.7,9.5,15.2,17.6};
    double * pointer = &arr[0];
    for(int i = 0; i < 7; i++)
    {
        cout<<*(pointer + i)<<“ ”;
    }
    return 0;}
```

Common Programming Errors

- Attempting to store address in a variable not declared as pointer
- Using pointer to access nonexistent array elements
- Forgetting to use bracket set, [ ], after delete operator
- Initialized pointer variables incorrectly

Common Programming Errors (continued)

- When an address is required, any of the following can be used:
  - A pointer variable name
  - A pointer argument name
  - A non-pointer variable name preceded by the address operator
  - A non-pointer variable argument name preceded by the address operator

Summary

- Every variable has a data type, an address, and a value
- In C++, obtain the address of variable by using the address operator, &
- A pointer is a variable used to store the address of another variable
  - Must be declared
  - Use indirection operator, *, to declare the pointer variable and access the variable whose address is stored in pointer

Summary (continued)

- Array name is a pointer constant
- Arrays are passed to functions as addresses
- When a one-dimensional array is passed to a function, the function’s parameter declaration can be an array declaration or a pointer declaration
- Pointers can be incremented, decremented, compared, and assigned

Chapter 11: Introduction to Matlab

C++ FOR ENGINEERS
AND SCIENTISTS

Objectives

- In this chapter you will learn about:
  - What is Matlab?
  - Matlab Screen
  - Variables, array, matrix, indexing
  - Operators (Arithmetic, relational, logical)
  - Display Facilities
  - Flow Control
  - Using of M-File
  - Debugging

Variables

- No need for types. i.e.,

~~int a;~~
~~double b;~~
~~float c;~~

- All variables are created with double precision unless specified and they are matrices.

Example:

```
>>x=5;
>>x1=2;
```

- After these statements, the variables are 1x1 matrices with double precision

Workspace

- The workspace is Matlab’s memory
- Can **manipulate variables** stored in the workspace

```
>> a=12;
>> b=10;
>> c=a+b
c =
   22
>>
```

Variables

- **View variable contents by simply typing the variable name at the command prompt**

```
>> a
a =
12
>>
    >> a*2
a =
24
>>
```

Array, Matrix

- A vector `x = [1 2 5 1]`

```
x =
 1   2   5   1
```

- A matrix `t = [1 2 3; 5 1 4; 3 2 -1]`

```
t =
 1     2     3
    5     1     4
    3     2    -1
```

- Transpose `y = x’` `y =`

```
   1
2
   5
1
```

The : operator

- VERY important operator in Matlab
- Means ‘to’

```
>> 1:10
ans =
   1    2    3    4    5    6    7    8    9   10
>> 1:2:10
ans =
   1    3    5    7    9
```

🔍 Try the following

```
>> x=0:pi/12:2*pi;
>> y=sin(x)
```

The : operator

```
>>A(3,2:3)
ans =
   1   7
>>A(:,2)
ans =
   2
   1
   1
```

```
A =
    3    2    1
    5    1    0
    2    1    7
```

🔍 **What'll happen if you type A(:,:) ?**

Long Array, Matrix

- `t = 1:10`

```
t =
 1    2    3    4   5   6    7   8    9    10
```

- `k = 2:-0.5:-1`

```
k =
 2   1.5   1   0.5   0   -0.5   -1
```

- `X = [1:4; 5:8]`

```
x =
 1       2       3      4
     5       6       7      8
```

Generating Vectors from functions

- **zeros(M,N) MxN matrix of zeros**

```
x = zeros(1,3)
x =
  0     0     0
```

- **ones(M,N) MxN matrix of ones**

```
x = ones(1,3)
x =
  1     1     1
```

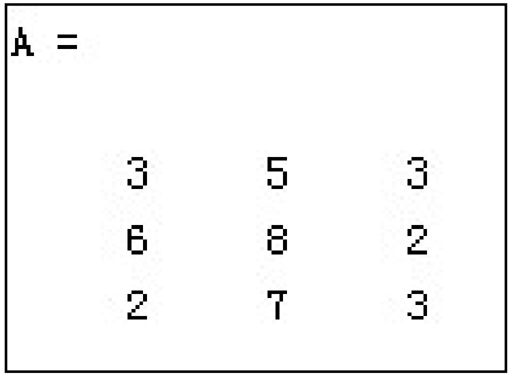
- **rand(M,N) MxN matrix of uniformly distributed random numbers on (0,1)**

```
x = rand(1,3)
x =
 0.9501  0.2311 0.6068
```

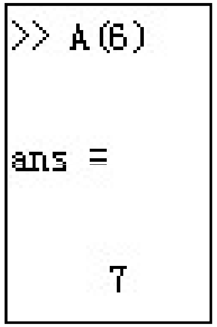
Matrix Index

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

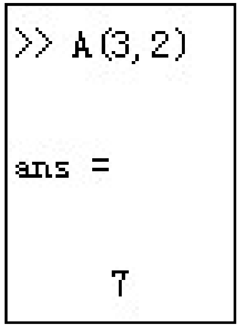
Given:



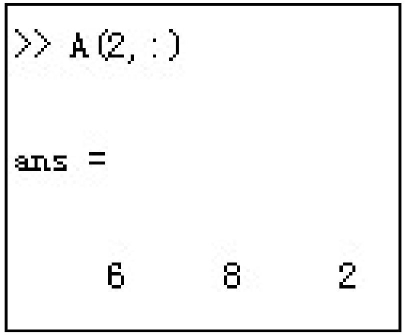
```
A =

     3     5     3
     6     8     2
     2     7     3
```



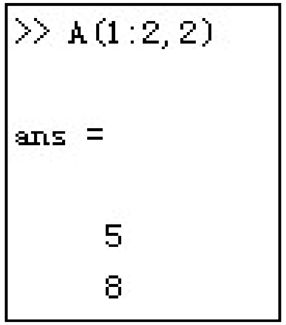
```
>> A(6)

ans =

     7
```



```
>> A(3,2)

ans =

     7
```



```
>> A(2,:)

ans =

     6     8     2
```



```
>> A(1:2,2)

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)

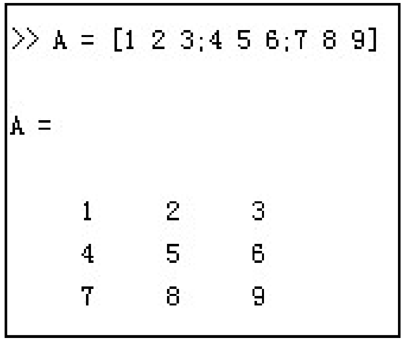
Error: ??? Index exceeds matrix dimensions.

Operators (arithmetic)

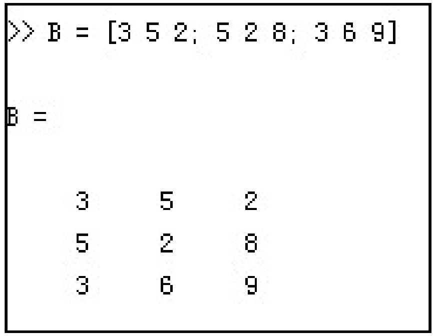
- + addition
- - subtraction
- * multiplication
- / division
- ^ power
- ‘ matrix transpose

Matrices Operations

Given A and B:

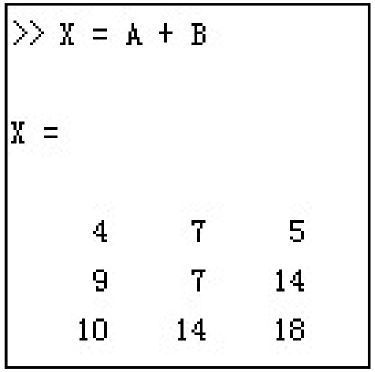


```
>> A = [1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```



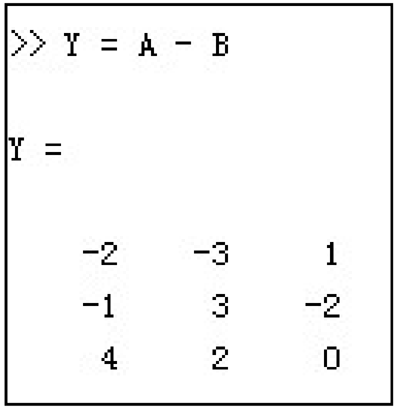
```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition



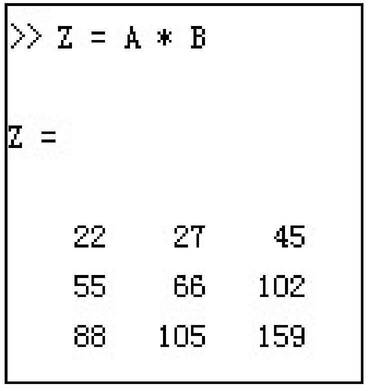
```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction



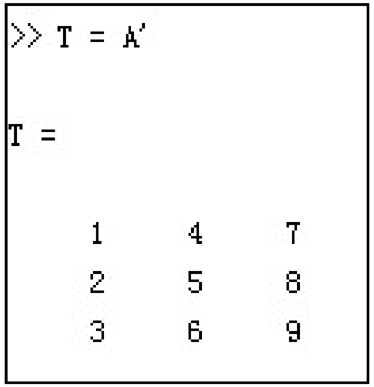
```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product



```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose



```
>> T = A'

T =

     1     4     7
     2     5     8
     3     6     9
```