Arrays

Mirza Mohammad Lutfe Elahi

Outline

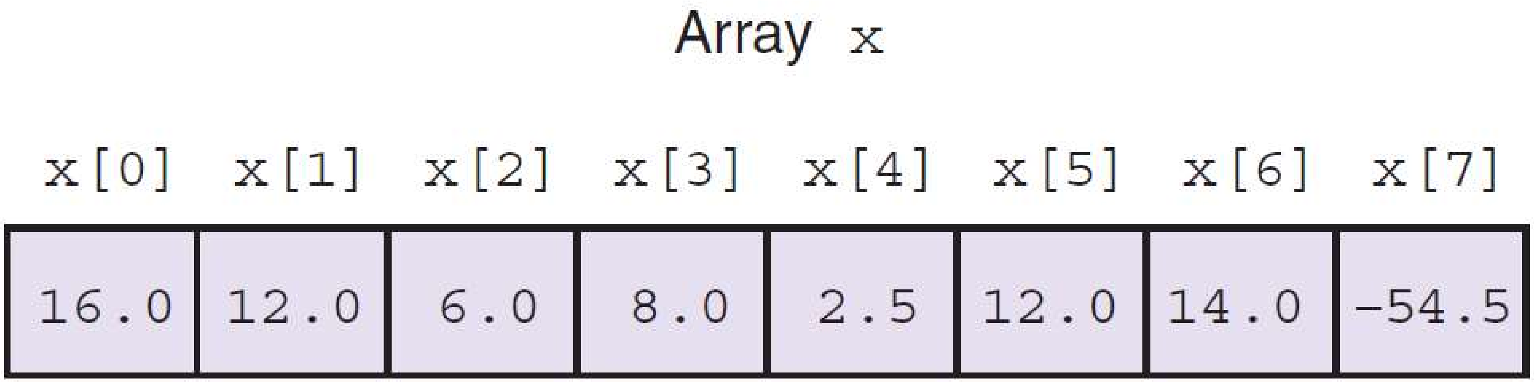
- Declaring, Initializing, and Indexing Arrays
- Using Loops for Sequential Array Access
- Using Array Elements as Function Arguments
- Array Arguments
- Partially Filled Arrays

What is an Array?

- **Scalar** data types, such as `int`, store a **single value**
- Sometimes, we need to store a collection of values
- An **array** is a collection of data items, such that:
  - All data values are of the **same type** (such as `int`)
  - Are referenced by the **same array name**
- Individual cells in an array are called **array elements**
- An array is called a **data structure**
  - Because it stores many data items under the same name
- Example: using an array to store exam scores

Declaring an Array

- To declare an array, we must declare:
  - The array **name**
  - The **type** of array element
  - The **number** of array elements
- Example: `double x[8];`
- Associate 8 elements with array name **x**



Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Initializing Arrays

- You can declare a variable without initialization

```
double average;  /* Not initialized */
```

- You can also declare a variable with initialization

```
int sum = 0;     /* Initialized to 0 */
```

- Similarly, you can declare arrays without initialization

```
double x[20];    /* Not initialized */
```

- You can also declare an array and initialize it

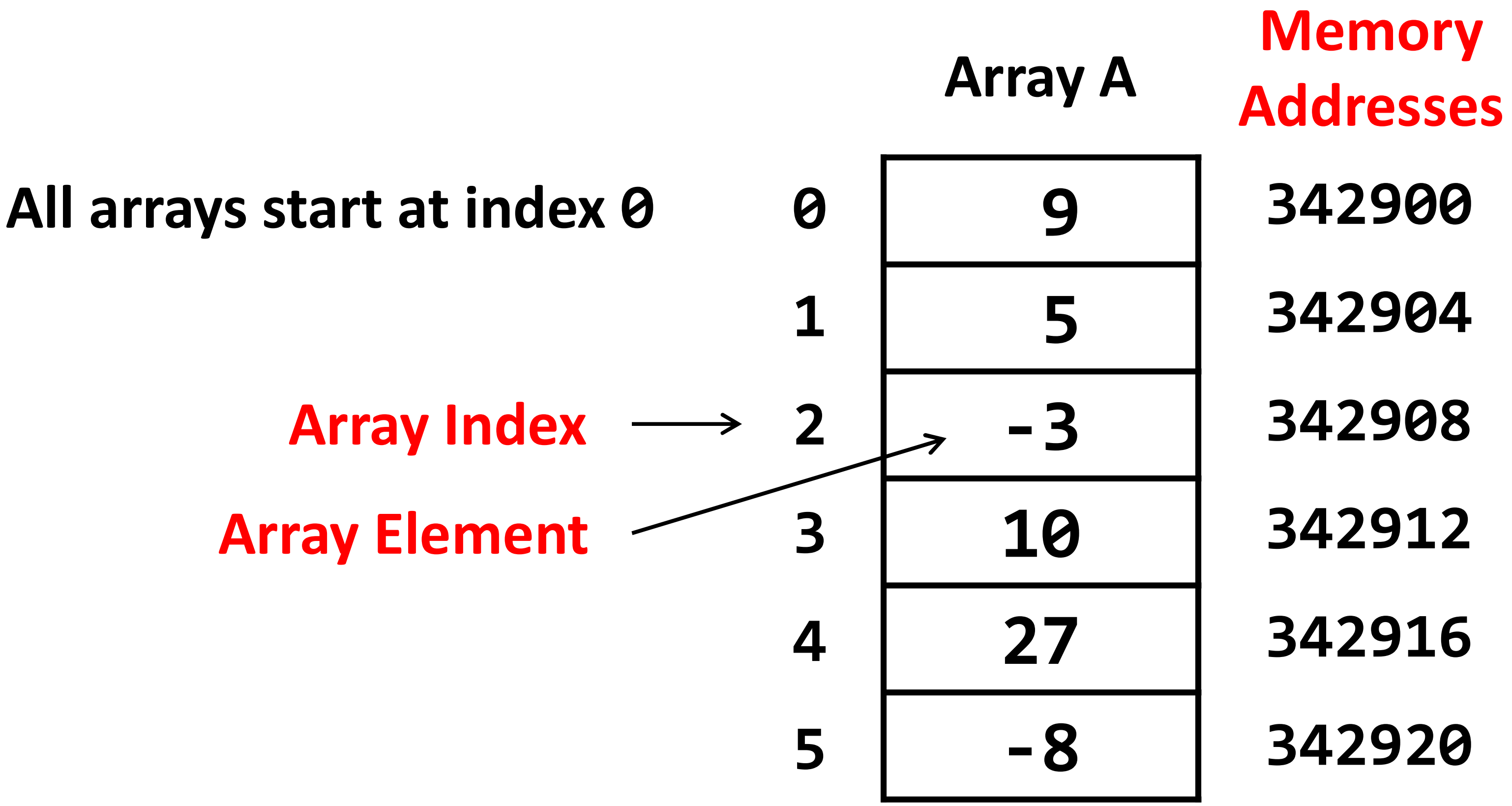
```
int prime[5] = {2, 3, 5, 7, 11};
```

- No need to specify the array size when initializing it

```
int prime[] =  {2, 3, 5, 7, 11};
```

Visualizing an Array in Memory

```
/* Array A has 6 elements */
int A[] = {9, 5, -3, 10, 27, -8};
```



All arrays start at index 0

Array Index

Array Element

	Array A	Memory Addresses
0	9	342900
1	5	342904
2	-3	342908
3	10	342912
4	27	342916
5	-8	342920

Array Indexing

```
double x[8];
```

- Each **element** of **`x`** stores a value of type **`double`**
- The elements are **`indexed`** starting with **index 0**
  - An array with **8 elements** is indexed from **0 to 7**
- **`x[0]`** refers to **0th element** (first element) of array **`x`**
- **`x[1]`** is the next element in the array, and so on
- The integer enclosed in brackets is the **array index**
- The index must range from **zero** to **array size – 1**

Array Indexing (Cont'd)

- An array **index** is also called a **subscript**
- Used to access individual array elements
- Examples of array indexing:

```
x[2] = 6.0;     /* index 2 */
y = x[i+1];     /* index i+1 */
```

- Array index should be any expression of type **`int`**
- A valid index must range from **0** to **array size – 1**
- C compiler does not provide array bound checking
- It is your job to ensure that each index is valid

Statements that Manipulate Array x

Array `x`

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

Arrays of Characters

- You can declare and initialize a **char array** as follows:

```
char vowels[] = {'A','E','I','O','U'};
```

- You can also use a string to initialize a **char array**:

```
char string[] = "This is a string";
```

- It is better to use a named constant as the array size:

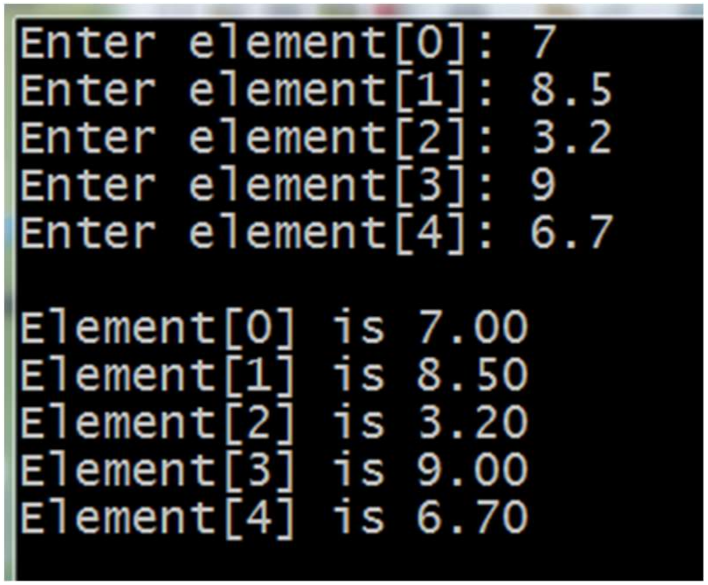
```
#define SIZE 100
. . .
char name[SIZE];  /* Not initialized */
```

- You can declare arrays and variables on same line:

```
char name[SIZE], answer;
```

Array Input/Output

```
#include<stdio.h>
#define SIZE 5    /* array size */
int main(void) {
   double x[SIZE];
   int i;

   for(i = 0; i < SIZE; i++) {
     printf("Enter element[%d]: ", i);
     scanf("%lf", &x[i]);
   }
   printf("\n");
   for(i = 0; i < SIZE; i++)
     printf("Element[%d] is %.2f\n", i, x[i]);

   return 0;
}
```



```
Enter element[0]: 7
Enter element[1]: 8.5
Enter element[2]: 3.2
Enter element[3]: 9
Enter element[4]: 6.7

Element[0] is 7.00
Element[1] is 8.50
Element[2] is 3.20
Element[3] is 9.00
Element[4] is 6.70
```

Computing Sum and Sum of Squares

```
/* We use a for loop to traverse an
 * array sequentially and accumulate
 * the sum and the sum of squares
 */

double sum = 0;
double sum_sqr = 0;

for(i = 0; i < SIZE; i++) {
  sum += x[i];
  sum_sqr += x[i] * x[i];
}
```

Computing Standard Deviation

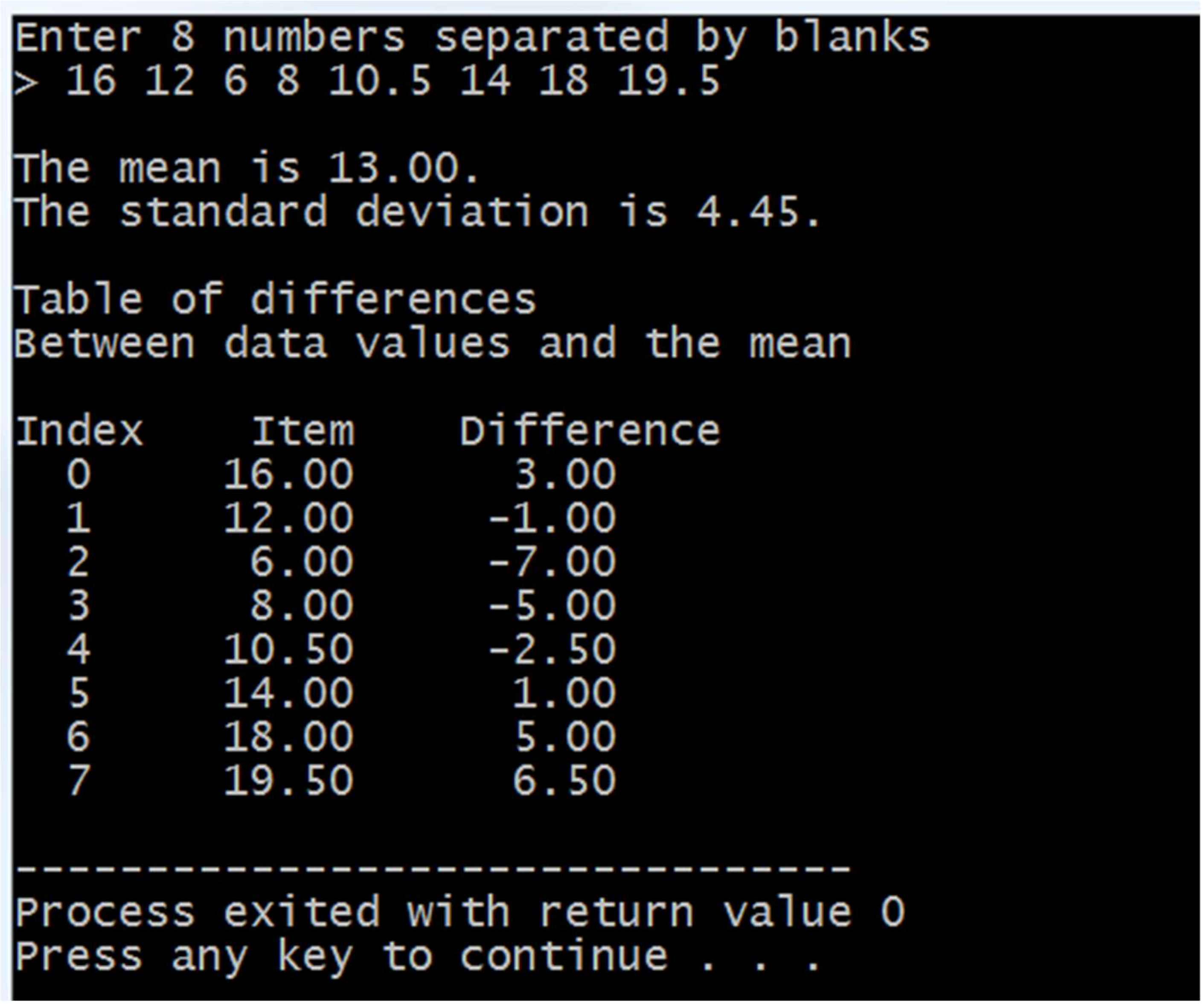
- The **mean** is computed as: **sum / SIZE**
- The Standard Deviation is computed as follows:

$$standard\ deviation = \sqrt{\frac{\sum_{i=0}^{SIZE-1} x[i]^2}{SIZE} - mean^2}$$

```
/* Program that computes the mean and standard deviation*/
#include <stdio.h>
#include <math.h>
#define SIZE  8    /* array size */

int main(void) {
  double x[SIZE], mean, st_dev, sum=0, sum_sqr=0;
  int i;

  /* Input the data */
  printf("Enter %d numbers separated by blanks\n> ", SIZE);
  for(i = 0; i < SIZE; i++) scanf("%lf", &x[i]);

  /* Compute the sum and the sum of the squares */
  for(i = 0; i < SIZE; i++) {
    sum += x[i];
    sum_sqr += x[i] * x[i];
  }
```

```
 /* Compute and print the mean and standard deviation */
  mean = sum / SIZE ;
  st_dev = sqrt(sum_sqr / SIZE – mean * mean);
  printf("\nThe mean is %.2f.\n", mean);
  printf("The standard deviation is %.2f.\n", st_dev);

  /* Display the difference between an item and the mean */
  printf("\nTable of differences ");
  printf("\nBetween data values and the mean\n\n");
  printf("Index    Item    Difference\n");
  for(i = 0; i < SIZE; i++)
    printf("%3d %9.2f %9.2f\n", i, x[i], x[i] - mean);

  return 0;
}
```

Sample Run…



```
Enter 8 numbers separated by blanks
> 16 12 6 8 10.5 14 18 19.5

The mean is 13.00.
The standard deviation is 4.45.

Table of differences
Between data values and the mean

Index    Item    Difference
  0     16.00     3.00
  1     12.00    -1.00
  2      6.00    -7.00
  3      8.00    -5.00
  4     10.50    -2.50
  5     14.00     1.00
  6     18.00     5.00
  7     19.50     6.50


--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Array Elements as Function Arguments

- From the last example:

`x[i]` is used as an actual argument to `printf`

```
printf("%3d %9.2f %9.2f\n", i, x[i], x[i]-
mean);
```

- The value of `x[i]` is passed to `printf`
- Similarly, `&x[i]` was an actual argument to `scanf`

```
scanf("%lf", &x[i]);
```

- The address `&x[i]` is passed to `scanf`
- Array elements are treated as scalar variables

Array Elements as Function Arguments

- Suppose that we have a function **do_it** defined as:

```
void do_it(double arg_1, double *arg2_p, double
*arg3_p) {
  *arg2_p = …
  *arg3_p = …
}
```

- Let **x** be an array of **double** elements declared as:

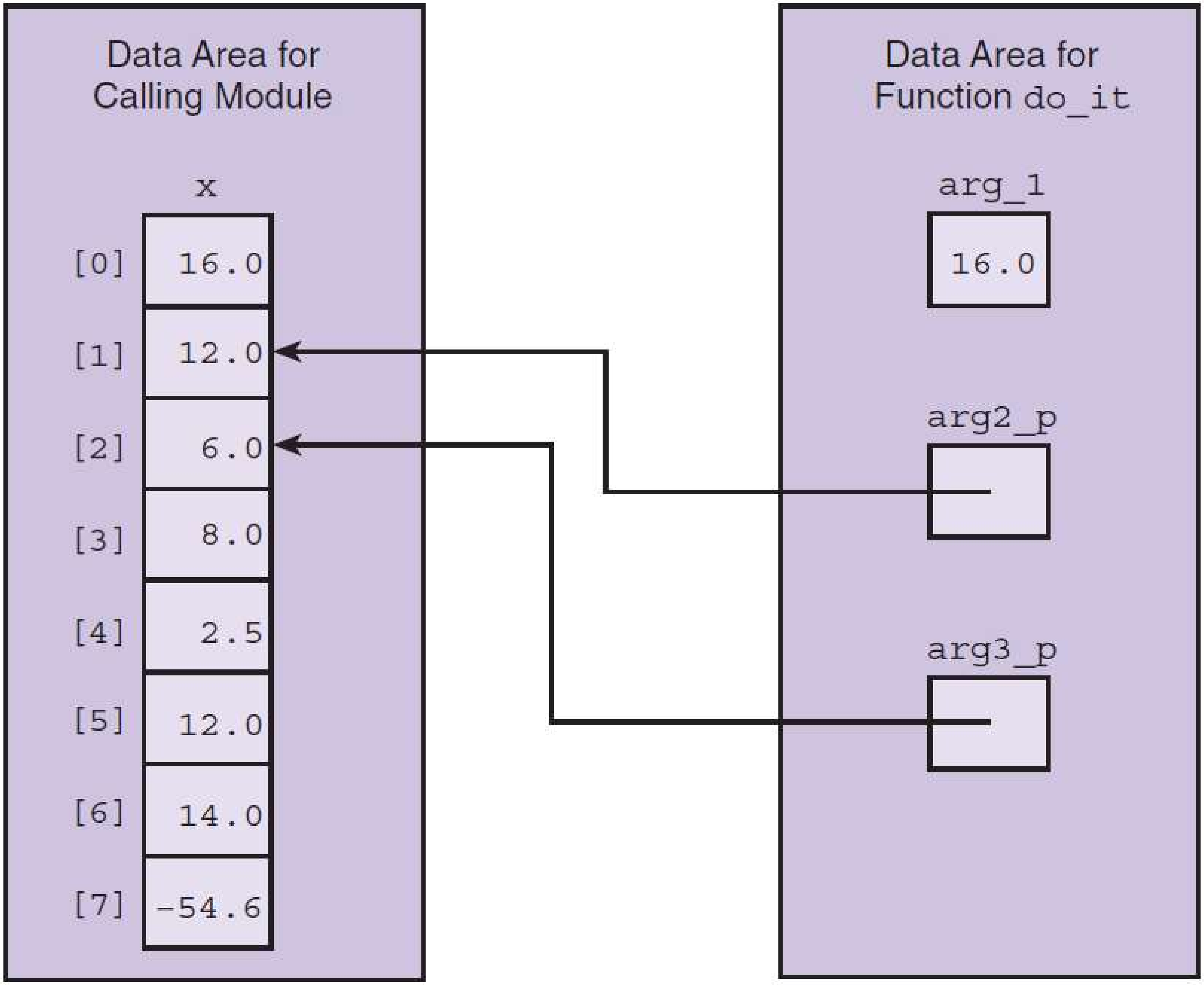
```
double x[8] = {16.0, 12.0, 6.0, 8.0, 2.5, 12.0,
14.0, -54.6};
```

- We can call the function **do_it** as follows:

```
do_it(x[0], &x[1], &x[2]);
```

- It will change the values of **x[1]** and **x[2]**

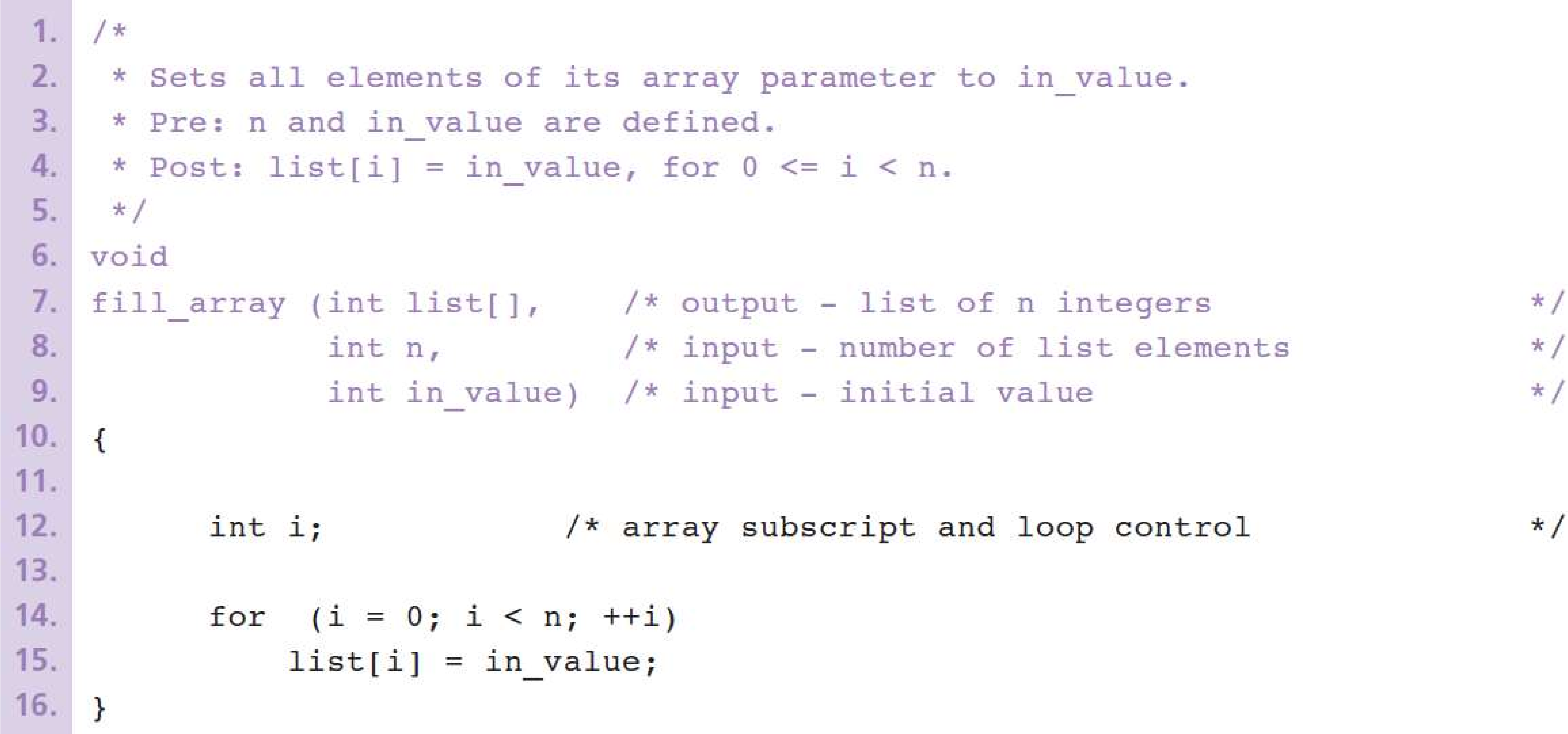
do_it(x[0], &x[1], &x[2]);


Data Area for
Calling Module
x
[0]
16.0
[1]
12.0
[2]
6.0
[3]
8.0
[4]
2.5
[5]
12.0
[6]
14.0
[7]
-54.6
Data Area for
Function do_it
arg_1
16.0
arg2_p
arg3_p

Array Arguments

- Besides passing array elements to functions, we can write functions that **have arrays as arguments**
- Such functions can compute some or all of the array elements
- Unlike scalar variables where we have the option of passing either the **value** or **address** of a variable to a function, **C only passes the address of an array** to a function array argument
- An array **cannot be passed by value** to a function

Array Arguments



```
/*
 * Sets all elements of its array parameter to in_value.
 * Pre: n and in_value are defined.
 * Post: list[i] = in_value, for 0 <= i < n.
 */
void
fill_array (int list[],    /* output - list of n integers                */
            int n,         /* input - number of list elements            */
            int in_value)  /* input - initial value                      */
{

      int i;            /* array subscript and loop control            */

      for  (i = 0; i < n; ++i)
          list[i] = in_value;
}
```

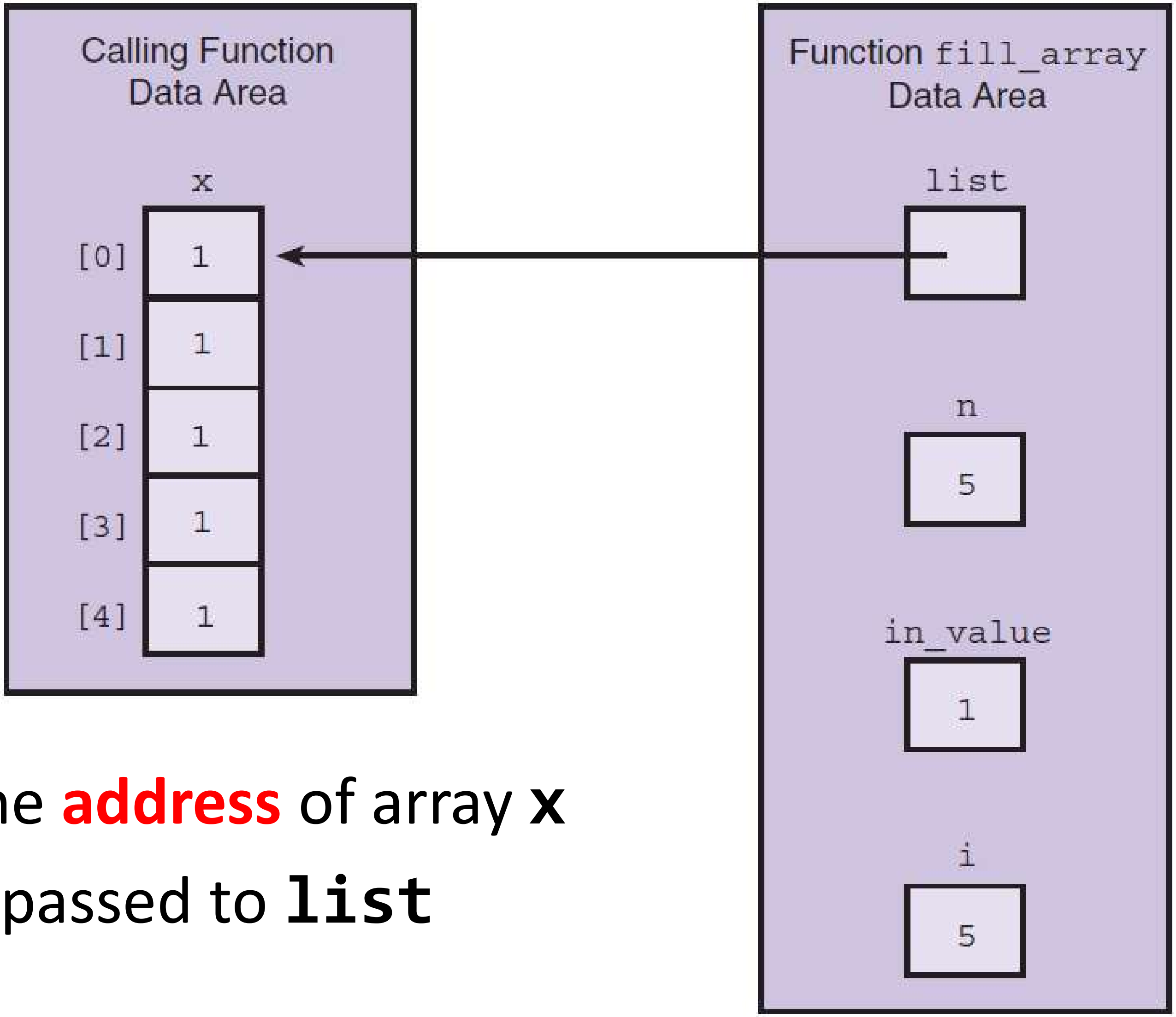
- **`list[]`** parameter does not specify the array size
- We can pass an array of **any size** to the function

Calling Function `fill_array`

- To call **`fill_array`**, you must pass 3 arguments:
  - Actual array name to fill
  - Number of array elements to fill
  - Value to store in array
- Examples of calling **`fill_array`**:

```
/* fill 5 elements of x with 1 */
fill_array(x, 5, 1);

/* fill 10 elements of y with num */
fill_array(y, 10, num);
```

`fill_array(x, 5, 1)`



The **address** of array **x**

is passed to **list**

An Array Argument is a Pointer

- Equivalent declarations of function **`fill_array`**

```
void fill_array(int list[], int n, int val);
void fill_array(int *list,  int n, int val);
```

The first declaration is more readable and preferable

- Equivalent calls to function **`fill_array`**

```
fill_array(x, 5, num);
fill_array(&x[0], 5, num);
```

The first call is more readable and preferable

Arrays as Input Arguments

The **const** keyword indicates that **list[]** is an input parameter that cannot be modified by the function

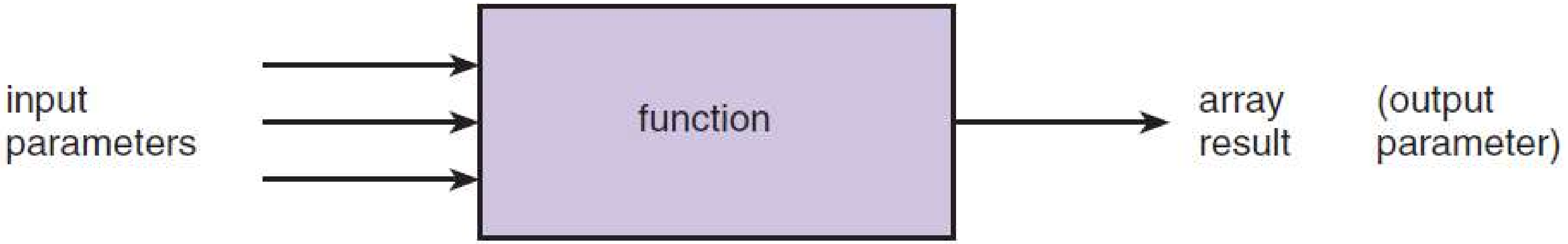
```
/* Returns the max in an array of n elements */
/* Pre: First n elements of list are defined */
double get_max(const double list[], int n) {
  int i;
  double max = list[0];

  for(i = 1; i < n; ++i)
    if(list[i] > max) max = list[i];

  return max;
}
```

Compute Average of Array Elements

```
/* Returns the average of n array elements */
/* Pre: First n elements of list are defined */
double get_average(const double list[], int n)
{
  int i;
  double sum = 0;
  for(i = 0; i < n; ++i)
    sum += list[i];
  return (sum/n);
}
```

The **const** keyword indicates that `list[]` is an input parameter that cannot be modified by the function

Returning an Array Result



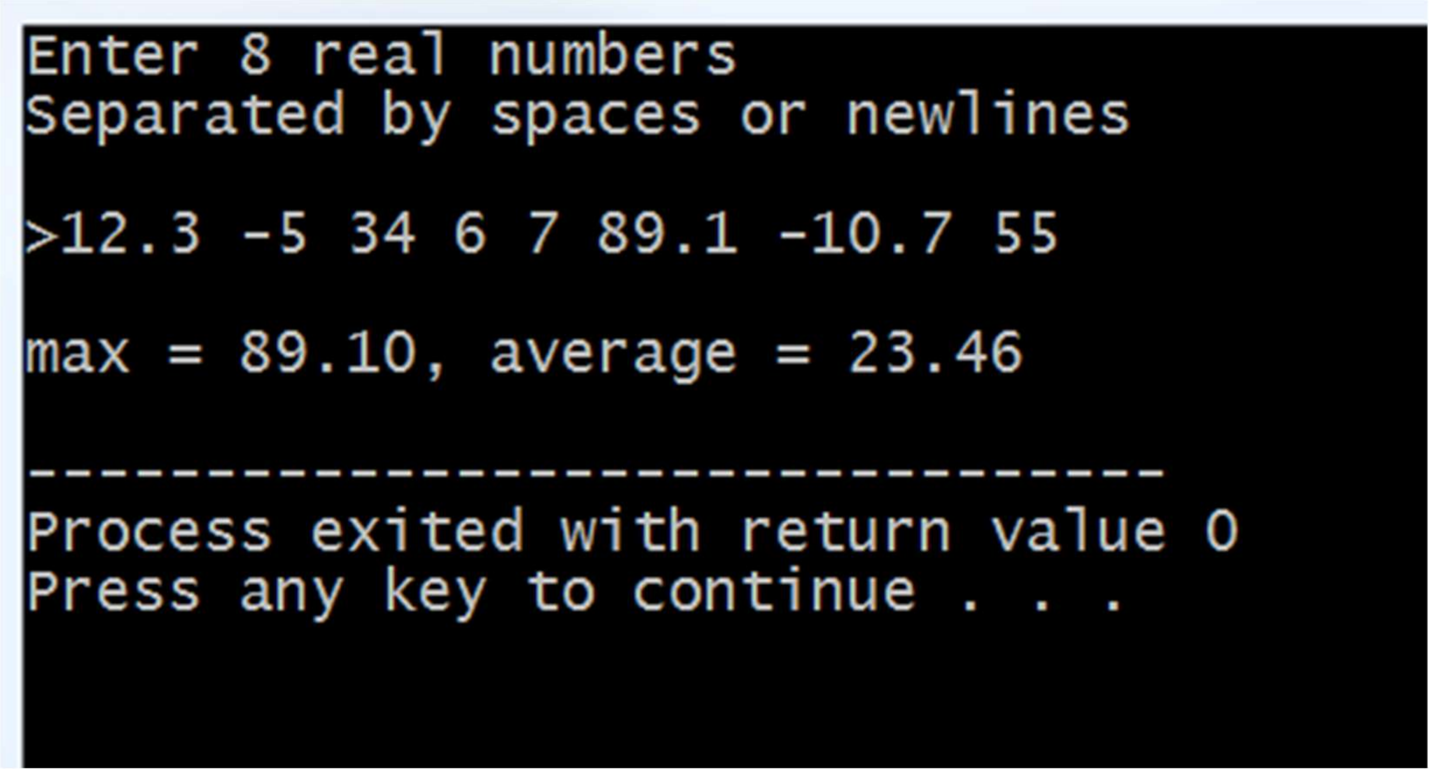
- In C, the return type of a function **cannot be an array**
- Thus, to return an array as result from a function, we can only have the array as an **output parameter**
- Recall that output parameters for a function are declared as pointers
- An array parameter is also a pointer
- Thus, an array parameter is an output parameter,   unless the **`const`** keyword is used

Example: read_array

```
/* read n doubles from the keyboard */
/* return an array of n doubles */

void read_array (double list[], int n) {
  int i;

  printf("Enter %d real numbers\n", n);
  printf("Separated by spaces or newlines\n");
  printf("\n>");

  for(i = 0;  i < n;  ++i)
    scanf("%lf", &list[i]);
}
```

```c
/* Program to compute max and average of an array */

#include <stdio.h>
#define  SIZE 8

void   read_array  (double list[], int n);
double get_max     (const double list[], int n);
double get_average (const double list[], int n);

int main(void) {
  double array[SIZE];

  read_array(array, SIZE);
  double max = get_max(array, SIZE);
  double ave = get_average(array, SIZE);

  printf("\nmax = %.2f, average = %.2f\n", max, ave);


  return 0;
}
```

Sample Run…



```
Enter 8 real numbers
Separated by spaces or newlines

>12.3 -5 34 6 7 89.1 -10.7 55

max = 89.10, average = 23.46

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Function to Add Two Arrays

```
/* Add n corresponding elements of arrays
   a[] and b[], storing result in array sum[] */
void
add_arrays(const double a[],  /* input array  */
           const double b[],  /* input array  */
           double sum[],      /* output array */
           int    n)          /* n elements   */
{
  int i;
  for(i = 0; i < n; i++)
    sum[i] = a[i] + b[i];
}
```

Partially Filled Arrays

- The format of array declaration requires that we specify the array size at the point of declaration
- Moreover, once we declare the array, its size cannot be changed. The array is a **fixed size** data structure
- There are many programming situations where we do not really know the array size before hand
- For example, suppose we want to read test scores from a data file and store them into an array, we do not know how many test scores exist in the file.
- So, what should be the array size?

Partially Filled Arrays (Cont'd)

- One solution is to declare the array big enough so that it can work in the worst-case scenario
- For the test scores data file, we can safely assume that no section is more than **`50`** students
- We define the **`SIZE`** of the array to be **`50`**
- However, in this case, the array will be partially filled and we cannot use **`SIZE`** to process it
- We must keep track of the actual number of elements in the array using another variable

Read an Array from a File

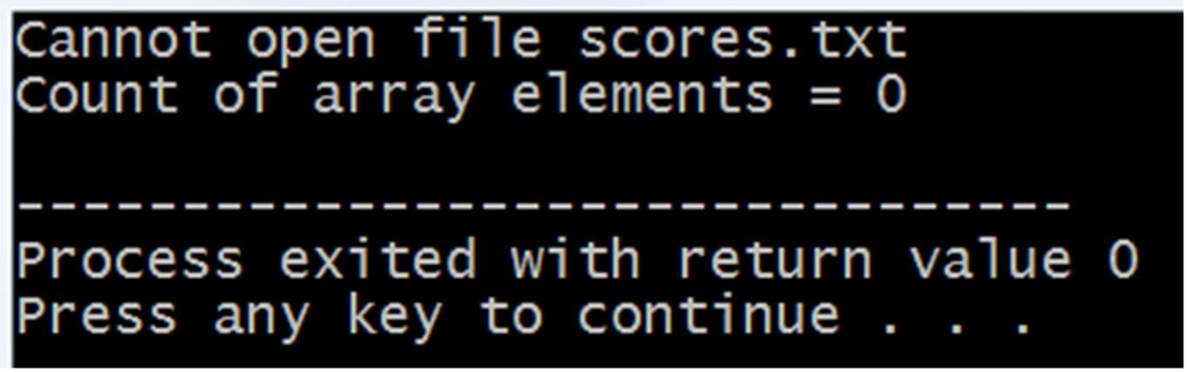
```c
#include <stdio.h>
#define  SIZE 50    /* maximum array size */

int  read_file(const char filename[], double list[]);
void print_array(const double list[], int n);

int main(void) {
  double array[SIZE];
  int count = read_file("scores.txt", array);
  printf("Count of array elements = %d\n", count);
  print_array(array, count);

  return 0;
}
```

```
int read_file(const char filename[], double list[]) {
  int count = 0;
  FILE *infile = fopen(filename, "r");

  if (infile == NULL) {  /* failed to open file */
    printf("Cannot open file %s\n", filename);
    return 0;                  /* exit function */
  }

  int status = fscanf(infile, "%lf", &list[count]);
  while (status == 1) {      /* successful read */
    count++;                   /* count element */
    if (count == SIZE) break;     /* exit while */
    status = fscanf(infile, "%lf", &list[count]);
  }

  fclose(infile);
  return count;      /* number of elements read */
}
```

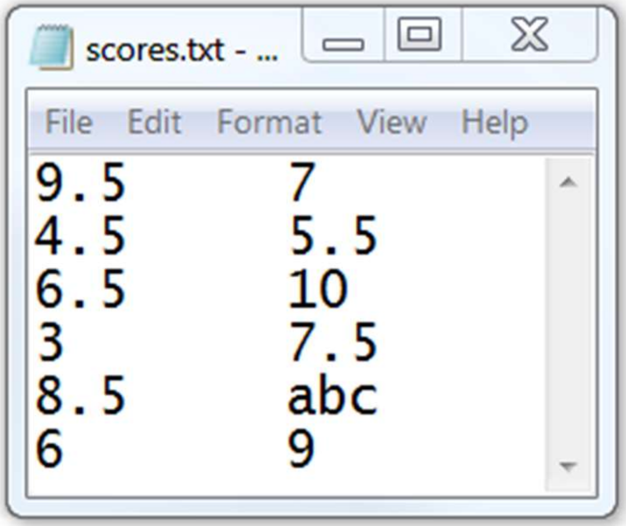
Function to Print an Array

```
void print_array(const double list[], int n) {
  int i;

  for (i = 0; i < n; i++)
    printf("Element[%d] = %.2f\n", i, list[i]);
}
```

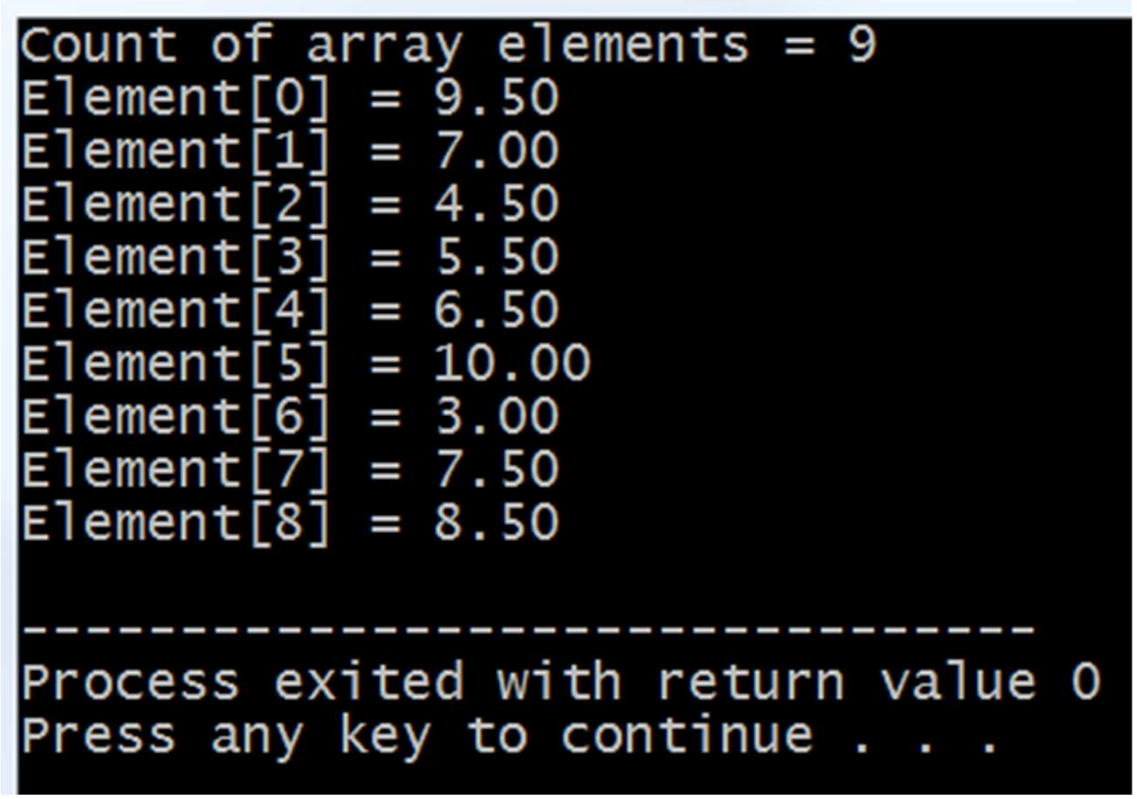
Sample Run



```
Cannot open file scores.txt
Count of array elements = 0

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```



```
9.5     7
4.5     5.5
6.5     10
3       7.5
8.5     abc
6       9
```



```
Count of array elements = 9
Element[0] = 9.50
Element[1] = 7.00
Element[2] = 4.50
Element[3] = 5.50
Element[4] = 6.50
Element[5] = 10.00
Element[6] = 3.00
Element[7] = 7.50
Element[8] = 8.50

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Cannot read
abc as **double**