



ICS-103

Computer Programming in C

Chapter 7: Arrays

Dr. Tarek Ahmed Helmy El-Basuny

Outline of Ch. 07 Topics

- What is an Array? and why do we use it?
- Declaring, Initializing, and Indexing one Dimensional Arrays
- Array Elements and Index
- Using Loops for Sequential Array Access with examples
- Statements that Manipulate Array x
- Declaring a string as an Array of Characters
- Using Array Elements as Function Arguments
- Function `fill_array`
- Passing an Array as Input Argument to the function
 - Program example to return the `max` in an array of `n` elements.
 - Program example to return the `average` of an array of `n` elements.
- Computing the sum of two Arrays
- Partially Filled Arrays and how to count the `#` of filled elements in the array.

What is an Array?

- ❑ So far, we were declaring **scalar** data types, such as **int**, **double** that each stores a **single value**. i.e. **int x;** or **char c;**
- ❑ **Sometimes**, we need to store a collection of values in one data type.
- ❑ An **array** is a collection of data items, such that:
 - All data values are of the **same type** (basic types, i.e. **int**, **char**, **double**,)
 - Stored contiguously in memory under one name
 - All data values are referenced by the **same array name with different indices**.
i.e. int A[10] //declares an array of **ten** integers with different indices.
 - **A[0], A[1], ..., A[9]****i.e. double B[20]** // declares an array of **twenty** long floating point numbers.
 - **B[0], B[1], ..., B[19]**
- ❑ An array is called a **data structure**.
 - Because it stores many data items under the same name
- ❑ A **one**-dimensional array is like a **list** while a **two**-dimensional **array** is like a **table**.
- ❑ Individual cells in an array are called **array elements**.

Why using an Array?

- ❑ "Assume we have a list of 1000 students' scores of a double type and we need to process them. If we are going to use the basic (scalar) data type (**double**), we will declare 1000 variable of type (**double**), like the following..."

```
int main(void)
{
    double studMark1, studMark2, studMark3, studMark4, ...,
        ..., studMark998, stuMark999, studMark1000;
    ...
    return 0;
}
```

- ❑ Can you imagine, how long we have to write the declaration part by using normal variable declaration?
 - By using an array, we can just declare like this in one statement:
 - **double studMark[1000];**
 - This will reserve 1000 contiguous memory locations for storing the students' scores and can be accessed by using the array name and different indices.

Declaring Arrays

- ❑ To declare **A single or one dimensional array**, the syntax is:
 - ➔ `array_element_data_type array_name[array_size];`
 - ➔ The `array_element_type`: is the type of data in the array
 - ➔ The `array_name`: you need to follow the rules of naming identifiers.
 - ➔ The `array_size` is the **number** of array elements.
- ❑ Example: `double x[8];`
- ❑ Associate **8** elements of type **double** with array name **x**.
- ❑ **We can declare multiple arrays of same type by using comma separated list, like regular variables: i.e. `int b[100], x[27];`**
- ❑ Array size may be determined explicitly (**Static**) or implicitly (**automatic**), i.e.

```
int A[13];           //static
#define CLASS_SIZE 73
double B[CLASS_SIZE]; //automatic
const int nElements = 25;
float C[nElements]; //automatic
```

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 in one step.**

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

- ❑ If size omitted, initializers determine it.

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

⇒ 5 initializes, therefore 5 element array

- ❑ You may not need to specify the array size when initializing it

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

- An array of an unknown number of integers (**allowable in a parameter of a function**)
- `C[0], C[1], ..., C[max-1]` //the size depends of the value of max.

Array Initialization

□ `int A[5] = {2, 4, 8, 16, 32};`

- Static

□ `int B[20] = {2, 4, 8, 16, 32};`

- Unspecified elements are guaranteed to be zero

□ `int C[4] = {2, 4, 8, 16, 32};`

- **Error** — compiler detects too many initial values

□ `int D[5] = {2*n, 4*n, 8*n, 16*n, 32*n};`

- After knowing `n`, the array elements initialized to expression's value.

□ `int E[n] = {1};`

- After knowing `n`, `E[0]` element initialized to `1`;
- All other elements initialized to `0`.

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 (subscript) →

Array Element's value

	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];` // `x` is an array of type `double` and of size `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 Element:** May be used wherever a variable of the same type may be used: i.e.
 - In an expression (including arguments)
 - On left side of assignment

- ❑ **Examples:**

↻ `A[3] = x + y; or`

↻ `x = y - A[3];`

- ❑ **Array index should be any expression of type `int`**
- ❑ `C` compiler does not provide array bound checking.
- ❑ It is your job to ensure that each index is valid.

Array Elements

□ Array elements are commonly used in loops

□ Examples,

- `for (i=0; i < max; i++)
A[i] = i*i;`
- `sum = 0;`
`For (j=0; j < max; j++)`
`sum += B[j];`
- `count=0;`
`for (rc!=EOF; count++)`
`rc=scanf ("%f", &A[count]);`

Caution! Caution!

- ❑ C does **NOT** check array bounds:
 - **I.e.**, whether an index points to an element within the array or not.
 - Might be high (**beyond the end**) or negative (**before the array starts**).

- ❑ It is the **programmer's responsibility to avoid indexing off the end of an array**:
 - Likely to corrupt data
 - May cause a segmentation fault
 - Could expose system to a security hole!

Statements that Manipulate Array **x**

- If we declared the following array:

```
double x[8] = {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -54.5};
```

```
    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
------	------	-----	-----	-----	------	------	-------

- We can use the **elements of the array x** wherever a variable of the same type may be used, as following:

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

Arrays of Characters

- ❑ You can **declare** and **initialize** an array of **char** 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"; // size is 17
```

- ❑ **char label[10] = "Single";** //Declares an array that looks like

S	i	n	g	l	e	\0	0	0	0
---	---	---	---	---	---	----	---	---	---

- Where **3** array elements are currently **unused**.
- Null character '**\0**' terminates strings.
- ❑ It is better to use a named constant as the array size so that once it has been changed the size of the array will be changed.

```
#define SIZE 100
```

```
char name[SIZE]; /* Not initialized */
```

- ❑ You can declare arrays and variables on same line if they have the same type:

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

Examples Using Character Arrays

- ❑ Character arrays **can be used to declare strings.**

- A **string** is actually one-dimensional array of characters in C language.

- String **“first”** is really a static array of characters

- Character arrays can be initialized using string literals

```
char string1[] = "first";
```

- ❑ Don't forget that one character is needed to store the **null character** (**\0**), which indicates the end of the string.

- **string1** actually has **5** elements but **the size of the string1 array is 6.**

- It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', \0 };
```

- Can access individual characters

```
string1[ 3 ] is character 's'
```

Array Input and Output

```
#include<stdio.h>
#define SIZE 5      /* array size */
int main(void) {
    double x[SIZE]; /*one dim. Array*/
    int i;
    //Read the elements of the array
    for (i=0; i<SIZE; i++) {
        printf("Enter element[%d]: ", i);
        scanf("%lf", &x[i]);
    }
    //Print the elements of the array
    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 can 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 the mean & Standard Deviation

- We can use the sum and the sum of squares to calculate:
 - The **mean** is computed as: **sum** / **SIZE**
 - The **Standard Deviation** is computed as follows:

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

Computing the mean & Standard Deviation

```
/* 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];
    }
}
```

Computing the mean & Standard Deviation

```
/* 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 of Computing the mean & Standard Deviation

```
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

- We noticed in the last example that:
- The value of `x[i]` is passed to `printf` as an actual argument:
 - ⇒ `printf("%3d %9.2f %9.2f\n", i, x[i], x[i]-mean);`
- Similarly, `&x[i]` was an actual argument to `scanf`
 - ⇒ `scanf("%lf", &x[i]);`
- That means, **array elements or their addresses** are treated as scalar variables and can be passed as function arguments.

Array Elements as Function Arguments

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

```
void do_it(double x, double *p1, double *p2)
{
    *p1 = x + 5;
    *p2 = x * x;
}
```

- Let `y` be an array of `double` elements declared as:

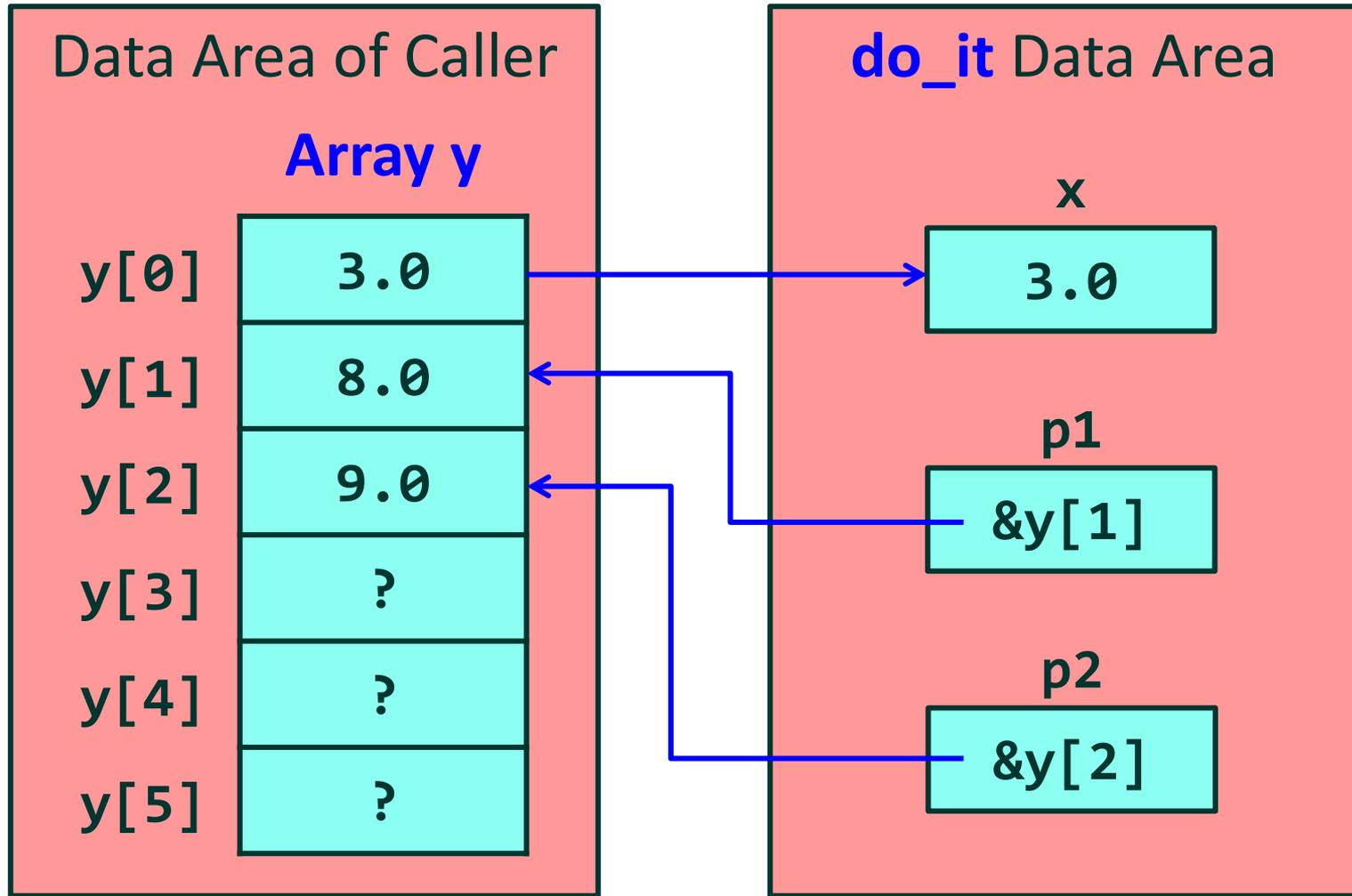
```
double y[6]; /* not initialized */
y[0] = 3.0; /* initialize y[0] to 3.0 */
```

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

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

- It will `change the values` of `y[1]` and `y[2]`

`do_it(y[0], &y[1], &y[2])`



Outline of Ch. 07 Topics

□ Last class, we discussed:

- What is an Array? and why do we use it?
- Declaring, Initializing, and Indexing one Dimensional Arrays
- Array Elements and Index
- Using Loops for Sequential Array Access with examples
- Statements that Manipulate Array x
- Declaring a string as an Array of Characters
- Using Array Elements as Function Arguments

□ Today's class, we are going to discuss:

- Function `fill_array`
- Passing an Array as Input Argument to the function
 - Program example to return the `max` in an array of `n` elements.
 - Program example to return the `average` of an array of `n` elements.
- Computing the sum of two Arrays
- Partially Filled Arrays and how to count the `#` of filled elements in the array.

Calling Function `fill_array`

- ❑ In C, we can fill an array by calling `fill_array` function as following: `fill_array(array name, Number of array elements to fill, Value to store in the array);`
- ❑ Notice, you must pass 3 arguments:
 - ➔ array name
 - ➔ Number of array elements
 - ➔ Value to store

- ❑ 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);
```

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 5 elements of x with num */*

➔ `fill_array(x, 5, num);`

➔ `fill_array(&x[0], 5, num);`

❑ The first call is more readable and preferable

Passing an Array as an Argument

- ❑ If you want to pass **a one-dimension array** “param” as an argument into a function, you would have to declare it as a formal parameter in one of following three ways and all three declaration methods produce similar results:
- ❑ Formal parameter as a sized array:
 - ➔ `void myFunction(int param[10]) { . . . }`
- ❑ Formal parameter as an unsized array:
 - ➔ `void myFunction(int param[]) { . . . }`
- ❑ Formal parameter as a pointer:
 - ➔ `void myFunction(int *param) { . . . }`
- ❑ **Example**: The function `getAverage` takes **an array as an input** along with another argument “size” and **returns the average** of the numbers passed through the array.

```
double getAverage(int arr[], int size) {  
    int i;  
    double avg;  
    double sum = 0;  
    for (i = 0; i < size; ++i) { sum += arr[i]; }  
    avg = sum / size;  
    return avg; }  
}
```

Example: Compute **max** and **average** of an array

```
/* 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() {  
    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;  
}
```

Example: read_array Elements

```
/* a function read_array reads n doubles from the  
keyboard and returns 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]);  
}
```

Compute **Max** in An Array (**Input Argument**)

- ❑ The **qualifier const** can be applied to the declaration of any variable to specify that its value will not be changed after initialization.
- ❑ The **const** keyword indicates that **list[]** is an input parameter **that cannot be modified by the function**.

```
/* Assume: n elements of an array list are defined */  
/* a function get_max returns the max in an array of  
n elements */
```

```
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

```
/* Assume: n elements of an array list are defined */  
/* a function get_average returns the average of n  
array elements */
```

```
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.

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[],    /* Sum 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.
- ❑ 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?
 - One solution is to **declare the array big enough** so that it can work in the worst-case scenario.
 - We can safely assume that no section is more than 50 students.
 - **Then, 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**. i.e. get the average, as some elements are not filled!!
 - **We must keep track of the actual number of elements in the array using another variable.**

Program to Read an Array from a File

```
#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() {
    double array[SIZE];
    int count = read_file("scores.txt", array);
    printf("Count of array elements = %d\n", count);
    print_array(array, count);
    return 0;
}
```

Program to Read an Array from a File

```
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) // or while (status != EOF)
    { /* 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 */
}
```

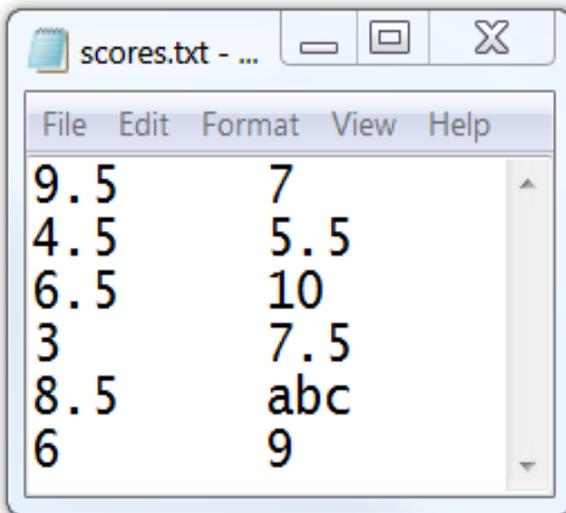
Function to Print an Array

```
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 . . .
```

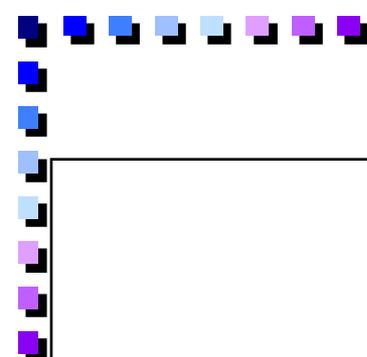


File	Edit	Format	View	Help
9.5		7		
4.5		5.5		
6.5		10		
3		7.5		
8.5		abc		
6		9		

```
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**



The End!!

Thank you

Any Questions?

