جامعة بيرزيت

BIRZEIT UNIVERSITY

COMPUTER SCIENCE DEPARTMENT FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER AND PROGRAMMING

Instructor :Murad Njoum

Programming in C

Ch7: Arrays

Objectives

- What is an Array?
- Declaring Arrays
- Visual representation of an Array
- Array Initialization
- Array Subscripts
- Accessing Array elements
- Using array elements as function arguments
- Using arrays as function arguments
- Returning an array result
- Partially filled Arrays
- Searching
- Sorting Introduction to 2-D Arrays
- Declaration of 2-D Arrays
- Accessing 2-D Array elements
- Initialization of 2-D Arrays
- Processing 2-D Arrays
- 2-D Arrays as parameters to functions

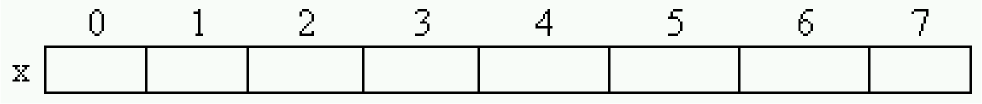
What is an Array?

- **Scalar** data types use a **single memory** cell to store a single value.
- For many problems you need to group data items together.
- A program that processes exam scores for a class, for example, would be easier to write if all the scores were stored in **one area of memory** and were able to be accessed as a group.
- C allows a programmer to group such related data items together into a single composite **data structure**.
- We now take a look at one such data structure: the **Array**.
- An **array** is a collection of two or more adjacent memory cells that:
  - Store the same type of data values (e.g. int)
  - Are referenced by the same name (i.e using one variable)
- These individual cells are called **array elements**

Declaring Arrays

- To declare an array, we must declare its name, type of data values it will store and the number of cells associated with it. Example:

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



- This instructs C to associate eight memory cells with the name x; these memory cells will be adjacent to each other in memory.
- You can declare arrays along with regular variables

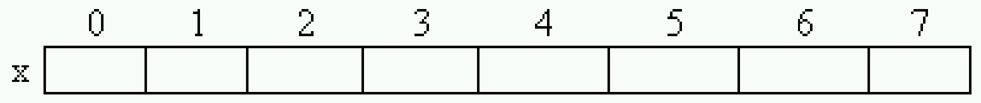
```
double cactus[5], needle, pins[7];
```

- It is a good practice to define the array size as constant:

```
#define ARRAY_SIZE  12
int myArray[ARRAY_SIZE];
```

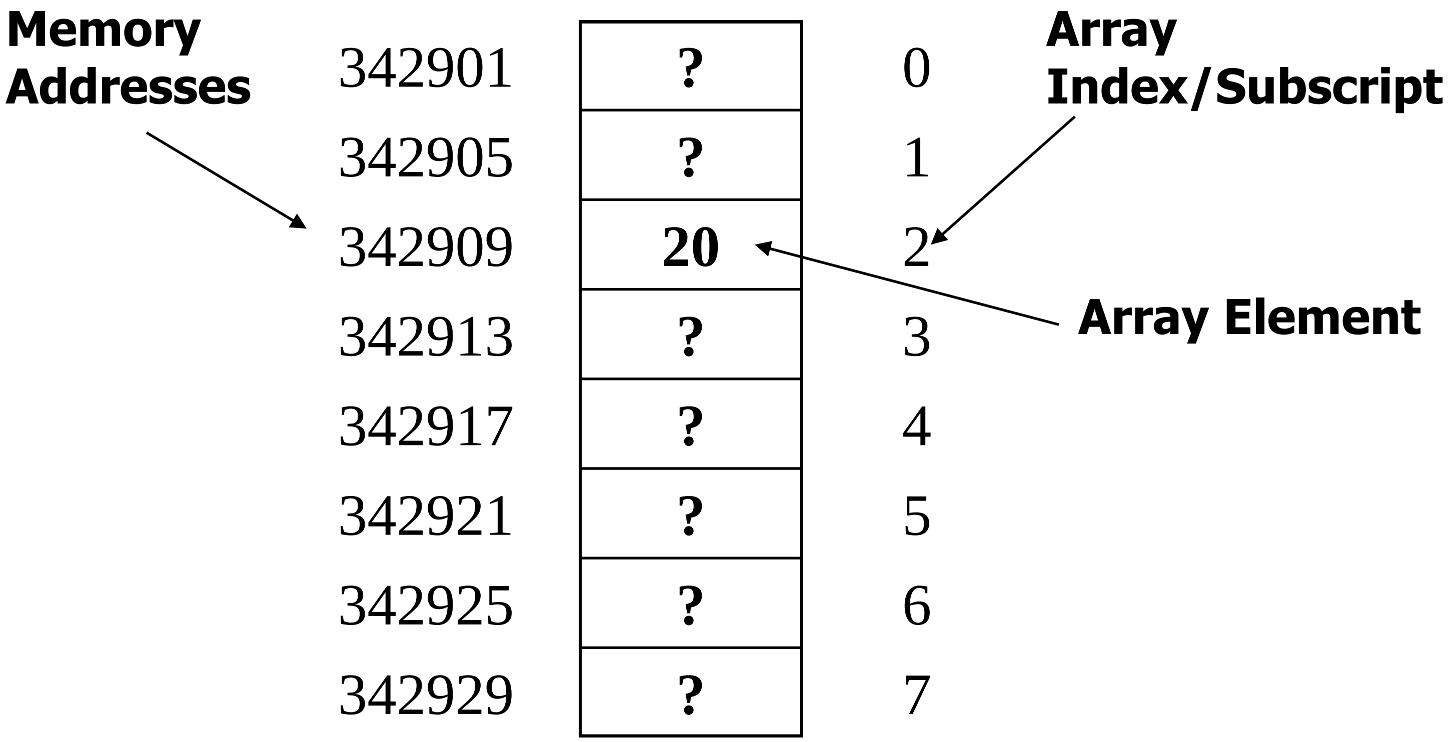
DECLARING ARRAYS …

- Each **element** of the array x may contain a single value of type double, so a total of eight such numbers may be stored and referenced using the array name **x**.
- **The elements are numbered starting with 0**
  - An array with 8 elements has elements at 0,1,2,3,4,5,6, and 7
- The subscripted variable **x[0]** (read as x sub zero) refers to the initial or **0th element** of the array x, x[1] is the next element in the array, and so on.
- The integer enclosed in brackets is the **array subscript** or **index** and its value must be in the range from zero to **one less** than the array size.



Visual representation of an Array

```
int x[8];  x[2] = 20;
```



Memory Addresses

Array Index/Subscript

Array Element

342901	?	0
342905	?	1
342909	**20**	2
342913	?	3
342917	?	4
342921	?	5
342925	?	6
342929	?	7

Note: Index starts with 0, not with 1

Array Initialization

- When you declare a variable, its value isn’t initialized unless you specify.

```
int sum;          // Does not initialize sum
int sum = 1;      // Initializes sum to 1
```

- Arrays, like variables, aren’t initialized by default

```
int X[10];  //creates the array, but doesn’t set any of its values.
```

- If you have all the values at the point of declaring the array, you can declare and initialize the array at the same time, like:

```
int X[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

- The array elements are initialized in the order listed

```
X[0] = 2
X[4] = 11
```

Array Initialization …

- If there are values in the initialization block, but not enough to fill the array, all the elements in the array without values are initialized to **0 in the case of double or int**, and **NULL in the case of char.**

```
int scores[20]  = {0};          // all 20 elements are initialized to 0
int scores[20]  = {1, 2, 3};  // First 3 elements are initialized to 1, 2,
                                 // 3 and the rest are initialized to 0
```

- If there are values in the initialization block, an explicit size for the array does not need to be specified. Only an empty array element is sufficient, C will count the size of the array for you.

```
int scores[]  = {20, 10, 25, 30, 40}; // size of the array score is
                                       // automatically calculated as 5
```

ARRAY SUBSCRIPTS

- We use subscripts/indices to differentiate between the individual array elements
- We can use any expression of type int as an array subscript.
- However, to create a valid reference, the value of this subscript must lie between 0 and one less than the array size.
- It is essential that we understand the distinction between an array subscript value and an array element value.

```
int x[2]; int y = 1; x[y] = 5;
```

The **subscript** is y (which is 1 in this case), and the **array element value** is 5

- C compiler does not provide any array bound checking. As a programmer it is your job to make sure that every reference is valid (i.e. it falls within the boundary of the array).

Accessing Array elements

1. point[1]   // the 2nd element of array *point* is accessed
2. point[9] = 20; // the 10th element of array point is assigned
                 // the value 20
3. We can use a loop to access all the elements of an array

   Example: Adding the values of all array elements

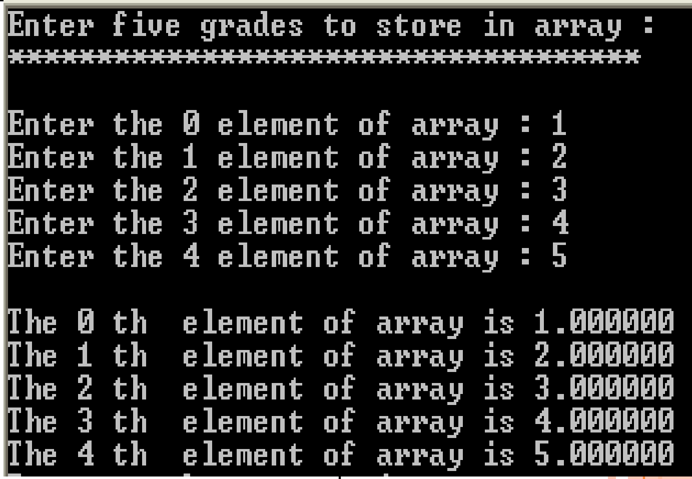
   Two alternative style for loops

```
    for ( i = 0; i < ARRAY_SIZE; i++)
              sum += a[i];
    for ( i = 0; i <= ARRAY_SIZE -1; i++)
           sum += a[i];
```

- Note : The array element is a single valued variable of the corresponding type and can be manipulated as a variable of that type.

Example 1

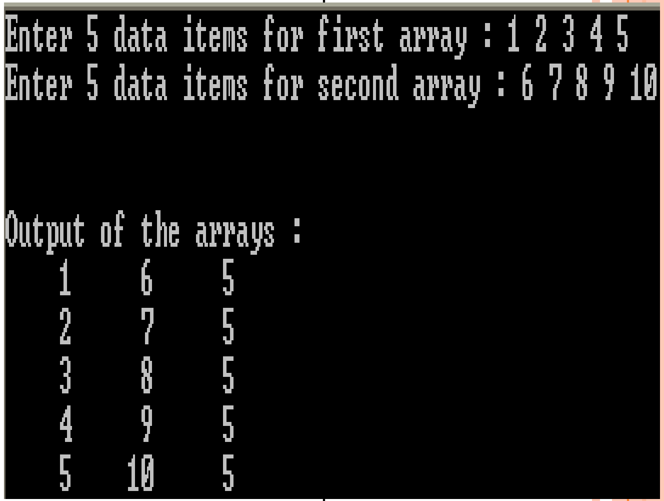
```
/* Reads five grades and print them */
#include<stdio.h>
#define SIZE 5

int main(void) {
    double  grades[SIZE] ; // array declaration
    int      i ;

    printf("Enter five grades to store in array : \n");
    printf("************************************\n\n");

    for (i = 0; i < SIZE; ++i) // loop to read the five grades into the array
    {
            printf ("Enter the %d element of array : ", i ) ;
            scanf ( "%lf", &grades[i] ) ;
    }

    printf("\n");
    for (i = 0; i < SIZE; ++i) // loop to display five grades stored in the
    array
            printf ("The %d th  element of array is %f\n", i, grades[i]) ;

    system("pause");
    return 0;
}
```



```
Enter five grades to store in array :
************************************

Enter the 0 element of array : 1
Enter the 1 element of array : 2
Enter the 2 element of array : 3
Enter the 3 element of array : 4
Enter the 4 element of array : 5

The 0 th  element of array is 1.000000
The 1 th  element of array is 2.000000
The 2 th  element of array is 3.000000
The 3 th  element of array is 4.000000
The 4 th  element of array is 5.000000
```

Example 2

```
/* Reads data into two arrays and subtract their corresponding elements,
   storing the result in another array. */
#include<stdio.h>
#define SIZE 5

int main(void) {
   int first[SIZE], second[SIZE], diff[SIZE], i;

   printf("Enter %d data items for first array : ", SIZE);
   for(i=0;i<SIZE; i++) // input first array
      scanf("%d", &first[i] );

   printf("Enter %d data items for second array : ", SIZE);
   for(i=0;i<SIZE; i++) // input second array
      scanf("%d", &second[i] );

   for(i=0;i<SIZE; i++)   // compute the differences
      diff[i]= second[i] - first[i];

   printf("\n\nOutput of the arrays : \n");
   for(i=0;i<SIZE; i++)   // output the arrays
      printf("%5d %5d %5d\n", first[i],  second[i],  diff[i] ) ;

   system("pause");
   return 0;
}
```



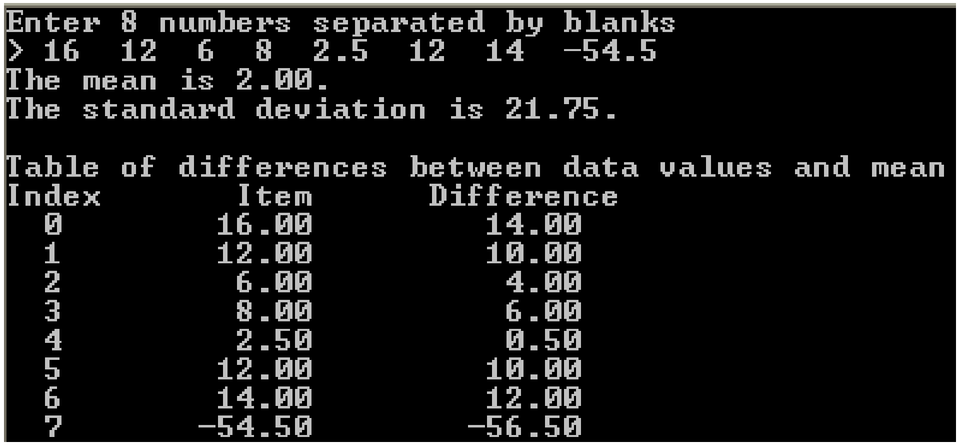
```
Enter 5 data items for first array : 1 2 3 4 5
Enter 5 data items for second array : 6 7 8 9 10


Output of the arrays :
    1     6     5
    2     7     5
    3     8     5
    4     9     5
    5    10     5
```

Example 3

```
/* Computes the mean and standard deviation of an array of data
   and displays the difference between each value and the mean.  */
#include <stdio.h>
#include <math.h>
#define SIZE  8  /* maximum number of items in list of data          */

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

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

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

Example 3…

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

 /* Displays the difference between each item and the mean */
  printf("\nTable of differences between data values and
mean\n");
  printf("Index       Item      Difference\n");
  for  (i = 0;  i < SIZE;  ++i)
    printf("%3d%4c%9.2f%5c%9.2f\n", i, ' ', x[i], ' ', x[i] - mean);

  system("pause");
  return (0);
}
```



```
Enter 8 numbers separated by blanks
> 16  12  6  8  2.5  12  14  -54.5
The mean is 2.00.
The standard deviation is 21.75.

Table of differences between data values and mean
Index       Item     Difference
  0        16.00         14.00
  1        12.00         10.00
  2         6.00          4.00
  3         8.00          6.00
  4         2.50          0.50
  5        12.00         10.00
  6        14.00         12.00
  7       -54.50        -56.50
```

USING ARRAY ELEMENTS AS FUNCTION ARGUMENTS

- From the last example, we note that **x[i]** is used as actual arguments to both *printf* and *scanf* functions.

```
for  (i = 0;  i < SIZE;  ++i)
        scanf("%lf", &x[i] );
```

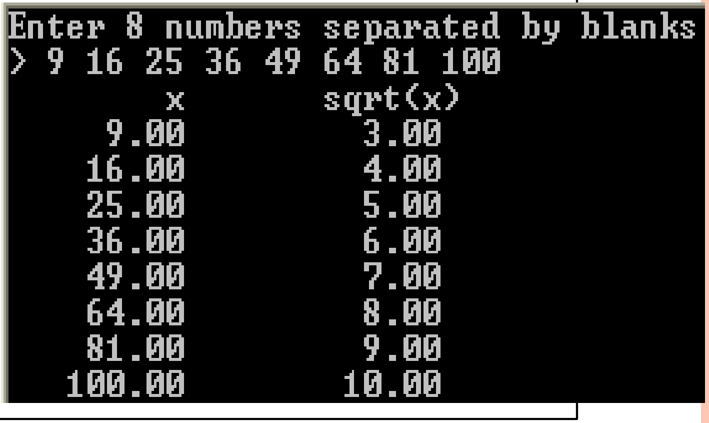
- When i is 3 for example, &x[i] in the above stement passes the address of the 4th array element , to the scanf function.

```
printf("%3d%4c%9.2f%5c%9.2f\n", i, ' ', x[i], ' ', x[i] - mean);
```

- Similarly, when i=3, x[i] in the above statement passes the value of the 4th element to the printf function.
- Thus, the important point to note is that array elements are treated as scalar variables and can be used wherever scalar variables can be used.
- For example, if we have a double array, `double a[8];  a[0] = 4.5;` Then we can make a function call: `double root = sqrt(a[0]);`
- This rule applies to all cases including user defined functions.

Example 4

```
/*  Finds the square roots of all elemets in an array */
#include <stdio.h>
#include <math.h>
#define SIZE  8  /* maximum number of items in list of data */

int main(void) {
     double x[SIZE], result[SIZE];
     int    i;

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

     /* Computes the square roots             */
     for   (i = 0;  i < SIZE;  ++i)
          result[i] = sqrt(x[i]);  /* notice the argument pass to sqrt function */

    /* print the result */
    printf("\tx\tsqrt(x)\n");
    for  (i = 0;  i < SIZE;  ++i)
          printf("%9.2f%4c%9.2f%\n", x[i], ' ', result[i]);

    system("pause");
    return 0;
 }
```

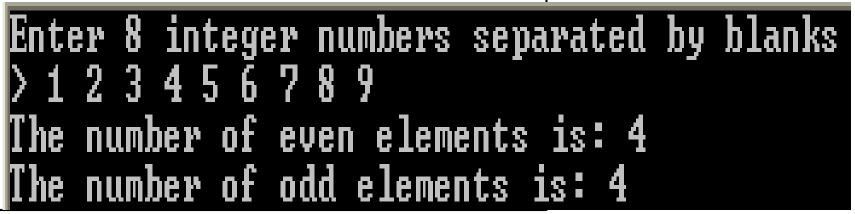


```
Enter 8 numbers separated by blanks
> 9 16 25 36 49 64 81 100
        x       sqrt(x)
     9.00         3.00
    16.00         4.00
    25.00         5.00
    36.00         6.00
    49.00         7.00
    64.00         8.00
    81.00         9.00
   100.00        10.00
```

Example 5

```
/*  uses a function to count even and odd numbers from an integer
    array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items in list of data        */

int is_even(int n);

int main(void) {
     int a[SIZE];
     int    i, evens = 0, odds = 0;

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

     /* counts the even and odd numbers      */
     for   (i = 0;  i < SIZE;  ++i) {
          if (is_even( a[i] ))   /* notice the argument pass to is_even
     function */
             evens++;
          else
             odds++;
    }
    printf("The number of even elements is: %d\n", evens);
    printf("The number of odd elements is: %d\n", odds);
    system("pause");
    return 0;
}
```

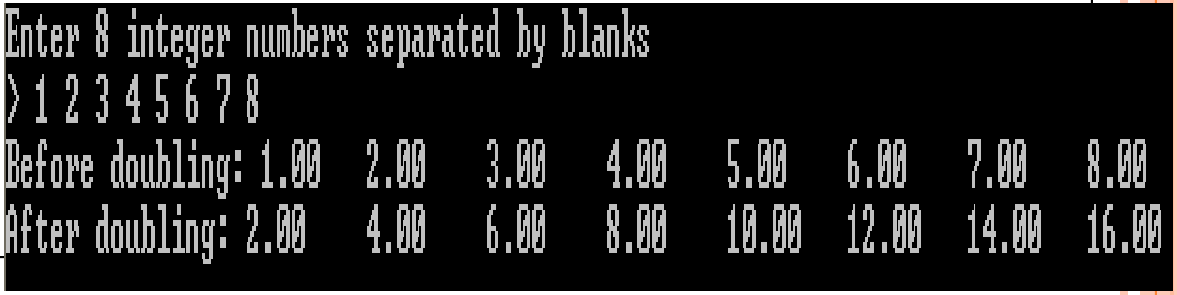
```
int is_even(int n) {
    return n%2 == 0;
}
```



```
Enter 8 integer numbers separated by blanks
> 1 2 3 4 5 6 7 8 9
The number of even elements is: 4
The number of odd elements is: 4
```

EXAMPLE 6

```
/*  Doubles each element of an array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items in list of data */
void doubler(double *x);
int main(void) {
     double x[SIZE];
     int    i;
     /* Gets the data  */
printf("Enter %d integer numbers separated by blanks\n> ", SIZE);
     for  (i = 0;  i < SIZE;  ++i)
            scanf("%lf", &x[i] );
  printf("Before doubling: ");
     for (i = 0;  i < SIZE;  ++i)
           printf("%.2f\t", x[i] );
        for (i = 0;  i < SIZE;  ++i)  doubler( &x[i] );
             printf("\nAfter doubling: ");
     for (i = 0;  i < SIZE;  ++i) printf("%.2f\t", x[i] );
     printf("\n");
     system("pause");
    return 0;
 }
 void doubler(double *x) {
     *x = *x * 2; }
```



```
Enter 8 integer numbers separated by blanks
> 1 2 3 4 5 6 7 8
Before doubling: 1.00   2.00    3.00    4.00    5.00    6.00    7.00    8.00
After doubling: 2.00    4.00    6.00    8.00    10.00   12.00   14.00   16.00
```

USING ARRAYS AS FUNCTION ARGUMENTS

- In addition to passing individual elements of an array to functions, we can also write functions that take an entire array as a single argument.
- Such functions can manipulate some or all of the array elements.
- However, unlike scalar variables where we have the option of either passing *value* or *reference* (address) of a variables to a function, C allows only passing by reference for arrays.
- In this section, we learn how to write functions that take array as argument and how to call such functions.

Array as formal parameter to functions

- To specify array as a formal parameter to a function, we put it as if we are declaring the array, but without specifying the size.

```
void print_array (int a[], …);
```

- Not specifying the size will allow the function to be called with any size of array.
- However, the function needs to know the size of the actual array in order to process it correctly. The solution is to add an extra parameter indicating the size or actual number of elements in the array.

```
void print_array(double a[], int size)
```

ARRAY AS ACTUAL ARGUMENT IN FUNCTION CALL

- To pass an array as actual argument to functions, we just give the array name without the brackets.

```
print_array (a, …);
```

- Since functions that take array usually also need to know the size of the array or at least the number of elements in the array, the complete call to the print_array function might be:

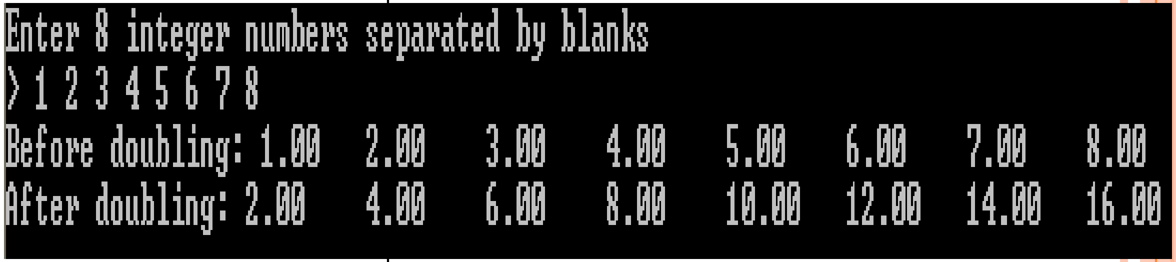
```
print_array(a, SIZE);
```

- Note that passing array as argument to functions is pass by reference not pass by value. Thus, no copy of the array is made in the memory area of the function. Instead, the function receives the address of the array and manipulates it indirectly.
- How does the function receive the address when we did not use & operator and the function did not declare a pointer variable?
- The truth is, array variables are in fact pointer variables, but which are declared differently.

Example 7

```
/*  Doubles each element of an array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items in list of data          */

void doubler(double *x);
void print_array(double a[], int size);

int main(void) {
      double x[SIZE];
      int    i;

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

      printf("Before doubling: ");
      print_array(x, SIZE);

      for (i = 0;  i < SIZE;  ++i)
           doubler( &x[i] );

      printf("After doubling: ");
      print_array(x, SIZE);
```

```
    system("pause");
    return 0;
}

void doubler(double *x) {
     *x = *x * 2;
}

void print_array(double a[], int size) {
      int i;

      for (i = 0;  i < size;  ++i)
            printf("%.2f\t", a[i]);
      printf("\n");
}
```

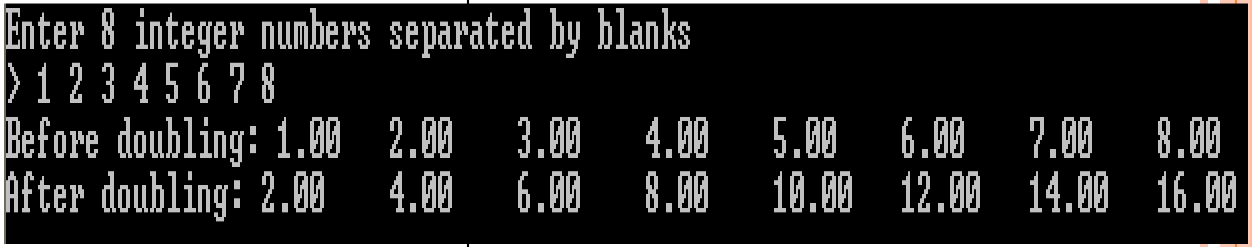


```
Enter 8 integer numbers separated by blanks
> 1 2 3 4 5 6 7 8
Before doubling: 1.00   2.00    3.00    4.00    5.00    6.00    7.00    8.00
After doubling: 2.00    4.00    6.00    8.00    10.00   12.00   14.00   16.00
```

Example 8

```
/*  Doubles each element of an array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items in
     list of data          */

void double_array(double a[], int size);
void print_array(double a[], int size);
int main(void) {
      double x[SIZE];
      int    i;

      printf("Enter %d integer numbers separated by
     blanks\n> ", SIZE);
      for  (i = 0;  i < SIZE;  ++i)
          scanf("%lf", &x[i] );
      printf("Before doubling: ");
      print_array(x, SIZE);
      double_array(x, SIZE);
      printf("After doubling: ");
      print_array(x, SIZE);

     system("pause");
     return 0;
}
```

```
void double_array( double a[], int
     size) {
     int i;

     for (i = 0;  i < size;  ++i)
          a[i] *= 2;
}

void print_array( double a[], int
     size)
 {
       int i;

       for (i = 0;  i < size;  ++i)
            printf("%.2f\t", a[i]);
       printf("\n");
 }
```

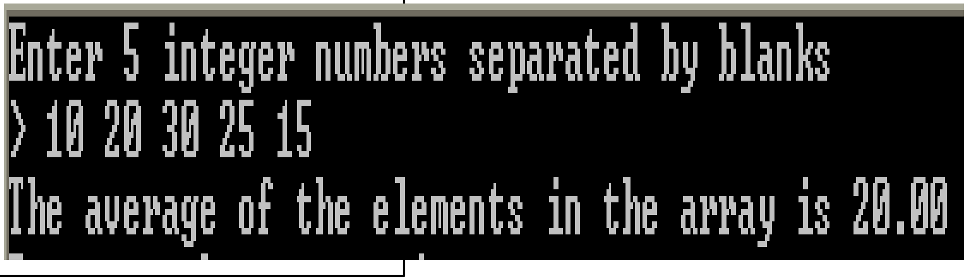


```
Enter 8 integer numbers separated by blanks
> 1 2 3 4 5 6 7 8
Before doubling: 1.00   2.00    3.00    4.00    5.00    6.00    7.00    8.00
After doubling: 2.00    4.00    6.00    8.00    10.00   12.00   14.00   16.00
```

Example 9

```
/*  Finds the average of elements in an array */
#include <stdio.h>
#define SIZE  5  /* maximum number of items
    in list of data     */

double get_average(double a[], int size);

int main(void) {
     double x[SIZE], average;
     int    i;

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

     average = get_average(x, SIZE);

     printf("The average of the elements in the
    array is %.2f\n", average);

    system("pause");
    return 0;
}
```

```
double get_average( double a[], int
size) {
    int i;
    double sum = 0;

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

    return sum/size;
}
```

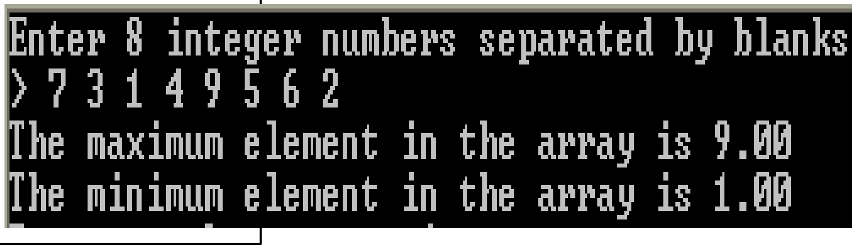


```
Enter 5 integer numbers separated by blanks
> 10 20 30 25 15
The average of the elements in the array is 20.00
```

Example 10

```
/*  Finds the maximum and minimum elements
    from an array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items
    in list of data     */
void get_max_min(double a[], int size, double
    *max, double *min);

int main(void) {
     double x[SIZE], maximum, minimum;
     int    i;

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

     get_max_min(x, SIZE, &maximum,
    &minimum);

     printf("The maximum element in the array
    is %.2f\n", maximum);
     printf("The minimum element in the array
    is %.2f\n", minimum);

    system("pause");
    return 0;
}
```

```
// uses output parameter to return
max & min
void get_max_min( double a[], int
size, double *max, double *min)
{
    int i;

    *max = a[0];
    *min = a[0];
    for (i = 1;  i < size;  ++i) {
         if (a[i] > *max)
           *max = a[i];
         else if (a[i] < *min)
           *min = a[i];
    }
}
```



```
Enter 8 integer numbers separated by blanks
> 7 3 1 4 9 5 6 2
The maximum element in the array is 9.00
The minimum element in the array is 1.00
```

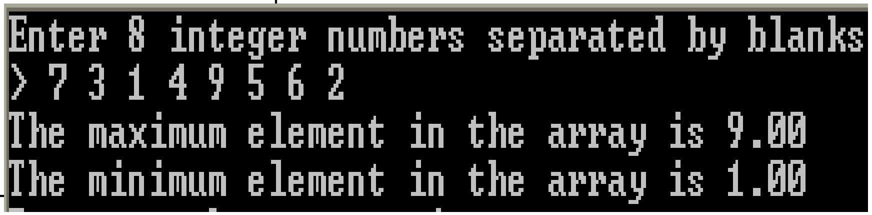
Returning an array result

- In C, the return type of a function cannot be an array.
- Thus, to return an array as result from a function, the only option is to use output parameter.
- We recall that output parameters for a function are declared as pointer variables.
- However, as mentioned earlier, an array variable is a pointer variable. Therefore formal parameters of type array are already output parameters.
- The next set of examples show various functions that return array as result:
  - Function, read_array, reads data from the user into an array and return it.
  - Function, add_arrays, uses two arrays as input, add the corresponding elements and return the result in another array.
  - Function, reverse_array, uses a single array for both input and output. It reverses the elements inside the array and return the reversed array.

Example 11

```
/*  Finds the maximum and minimum from an
    array .
   It uses a function that reads and returns an
    array */
#include <stdio.h>
#define SIZE  8  /* maximum number of items
    in list of data     */

void read_array(double a[], int size);
void get_max_min(double a[], int size, double
    *max, double *min);

int main(void) {
     double x[SIZE], maximum, minimum;

     read_array(x, SIZE);
     get_max_min(x, SIZE, &maximum,
    &minimum);

     printf("The maximum element in the array
    is %.2f\n", maximum);
     printf("The minimum element in the array
    is %.2f\n", minimum);

    system("pause");
    return 0;
}
```

```
void get_max_min(double a[], int size,
double *max, double *min) {
    int i;

    *max = a[0];
    *min = a[0];
    for (i = 1;  i < size;  ++i) {
         if (a[i] > *max)
           *max = a[i];
         else if (a[i] < *min)
           *min = a[i];
    }
}

void read_array (double a[], int size) {
    int i;
    printf("Enter %d real numbers
separated by blanks\n> ", size);
    for  (i = 0;  i < size;  ++i)
       scanf("%lf", &a[i]);
}
```

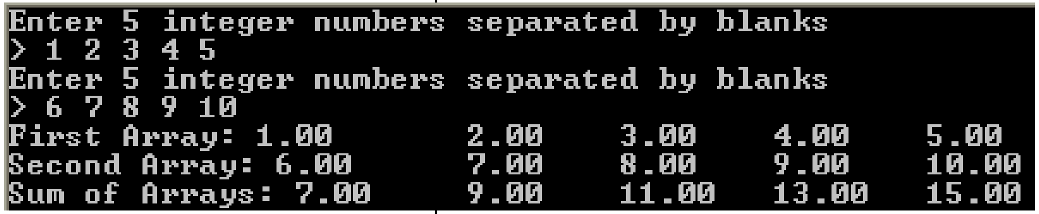


```
Enter 8 integer numbers separated by blanks
> 7 3 1 4 9 5 6 2
The maximum element in the array is 9.00
The minimum element in the array is 1.00
```

Example 12

```
/*  Adds two arrays and return the result in
    another array */
#include <stdio.h>
#define SIZE  5

void read_array(double a[], int size);
void print_array(double a[], int size);
void add_arrays(double a[], double b[], double c[],
    int size);

int main(void) {
    double first[SIZE], second[SIZE], sum[SIZE];

    read_array(first, SIZE);
    read_array(second, SIZE);
    add_arrays(first, second, sum, SIZE);

    printf("First Array: ");
    print_array(first, SIZE);
    printf("Second Array: ");
    print_array(second, SIZE);
    printf("Sum of Arrays: ");
    print_array(sum, SIZE);

    system("pause");
    return 0;
}
```

```
void read_array (double a[], int size) {
    int i;
    printf("Enter %d integer numbers
separated by blanks\n> ", size);
    for  (i = 0;  i < size;  ++i)
        scanf("%lf", &a[i]);
}
void add_arrays(double a[], double b[],
                          double c[], int size)
{
    int i;

    for (i=0; i<size; i++)
        c[i] = a[i] + b[i];
}
void print_array(double a[], int size) {
     int i;

     for (i = 0;  i < size;  ++i)
          printf("%.2f\t", a[i]);
     printf("\n");
}
```



```
Enter 5 integer numbers separated by blanks
> 1 2 3 4 5
Enter 5 integer numbers separated by blanks
> 6 7 8 9 10
First Array: 1.00       2.00    3.00    4.00    5.00
Second Array: 6.00      7.00    8.00    9.00    10.00
Sum of Arrays: 7.00     9.00    11.00   13.00   15.00
```

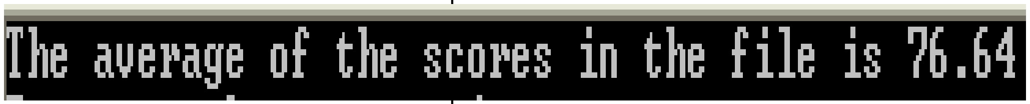
Partially filled Arrays

- The format of array declaration requires that we specify a size at the point of declaration.
- Moreover, once we decide on a size and declare the array, the size cannot be changed – array is fixed size data structure.
- There are many programming situations where we do not really know the number of elements before hand.
- For example, suppose we wish to read scores of students from a data file, store them into an array and then scan through the array to find the average.
- Obviously, we do not know how many scores are 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.
- For the scores data file, we can safely assume that no section is more than 50 students.
- However, in this case, the array will be partially empty and we cannot use SIZE in processing it. We must keep track of the actual elements in the array using another variable.

Example 13

```
/*  Finds the average score by reading scores from
     a data file  */
#include <stdio.h>
#define SIZE  50

double get_average(double a[], int size);

int main(void) {
     double x[SIZE], score, average;
     int status, count=0;
     FILE *infile;

     infile = fopen("scores.txt", "r");
     status = fscanf(infile, "%lf", &score);
     while (status != EOF) {
          x[count] = score;
          count++;
          status = fscanf(infile, "%lf", &score);
     }
     fclose(infile);
     average = get_average(x, count);
     printf("The average of the scores in the file is
     %.2f\n", average);

    system("pause");
    return 0;
}
```

```
double get_average(double a[], int size) {
    int i;
    double sum = 0;

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

    return sum/size;
}
```



```
The average of the scores in the file is 76.64
```

Searching

- Searching means scanning through a list of items (in an array) to find if a particular one exists.
- It usually requires the user to specify the target item – the item he wishes to locate
- If the target item is found, its location (index) is returned, otherwise, -1 is returned.

Linear Search Algorithm

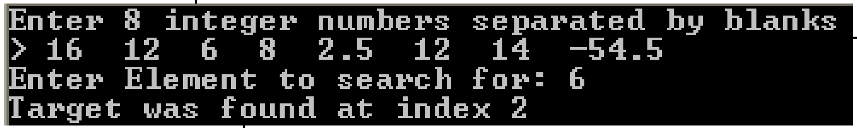
- This involves searching through the array sequentially until the target item is found or the array is exhausted.
- If the target is found, its location is returned, otherwise a flag such as –1 is returned. Here is the algorithm for Linear Search

```
1. Assume that the target has not been found
2. Start with initial array element
3. Repeat while the target is not found and there are more
   array elements
     4. If the current element matches the target
          5. Set a flag to indicate that the target has been found
        else
          6. Advance to the next array element
7. If the target was found
     8. Return the target index as the search result
   else
     9. Return -1 as the search result
```

Linear Search Implementation

```c
#include <stdio.h>
#define SIZE  8

int linear_search(double a[],  double target, int
    size);
void read_array(double a[], int size);

int main(void) {
     double x[SIZE], target;
     int index;

     read_array(x, SIZE);
     printf("Enter Element to search for: ");
     scanf("%lf", &target);
     index = linear_search(x, target, SIZE);
     if (index != -1)
        printf("Target was found at index %d\n",
    index);
     else
        printf("Sorry, target item was not found");
     system("pause");
     return 0;
}
void read_array (double a[], int size) {
    int i;
    printf("Enter %d integer numbers separated by
    blanks\n> ", size);
    for  (i = 0;  i < size;  ++i)
       scanf("%lf", &a[i]);
}
```

```c
/* Searches for target in an array using Linear
search;
 *  Returns index of target or -1 if not found  */

int linear_search(double a[],  double target,
                         int size)
{
     int i, found = 0,  where;

     i = 0;
     while (!found && i < size) {
        if (a[i] == target)
             found = 1;
        else
             ++i;
     }

     if (found)
          where = i;
     else
          where = -1;

     return where;
}
```



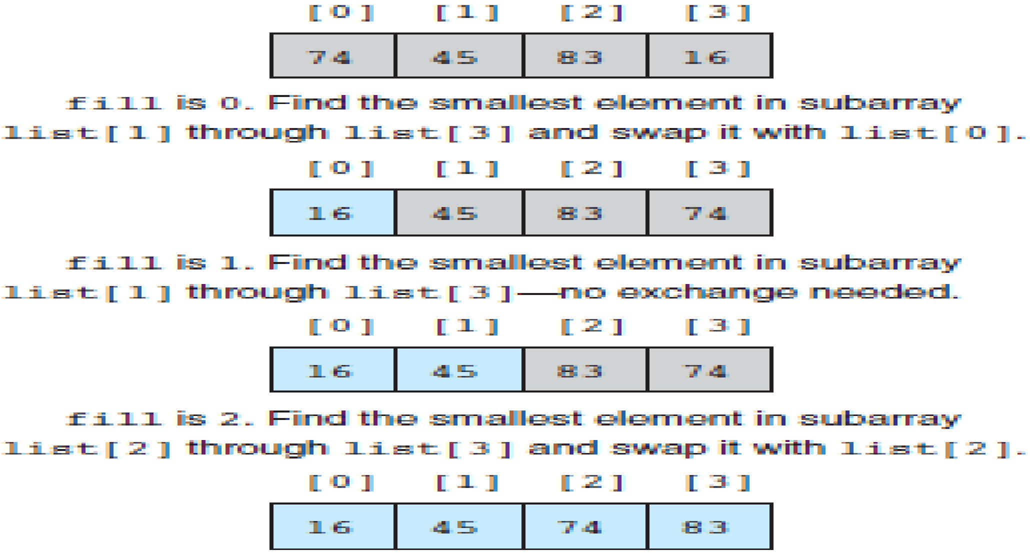
```
Enter 8 integer numbers separated by blanks
> 16   12   6   8   2.5   12   14   -54.5
Enter Element to search for: 6
Target was found at index 2
```

SORTING

- Sorting is the re-arrangement of a collection of data according to some key-field.
- It is a common activity in data management. Even when a list is maintained in a certain order, there is often a need to re-arrange the list in a different order.
- Because it takes so much processing time, sorting is a serious topic in computer science, and many different sorting algorithms have been designed.
- We shall consider one sorting method; Selection sort.

Selection Sort Algorithm

- Selection sort involved scanning through the list to find (or select) the smallest element and swap it with the first element.
- The rest of the list is then search for the next smallest and swap it with the second element.
- This process is repeated until the rest of the list reduces to one element, by which time the list is sorted.
- The following traces selection sort.



[0]	[1]	[2]	[3]
74	45	83	16

fill is 0. Find the smallest element in subarray list[1] through list[3] and swap it with list[0].

[0]	[1]	[2]	[3]
16	45	83	74

fill is 1. Find the smallest element in subarray list[1] through list[3]—no exchange needed.

[0]	[1]	[2]	[3]
16	45	83	74

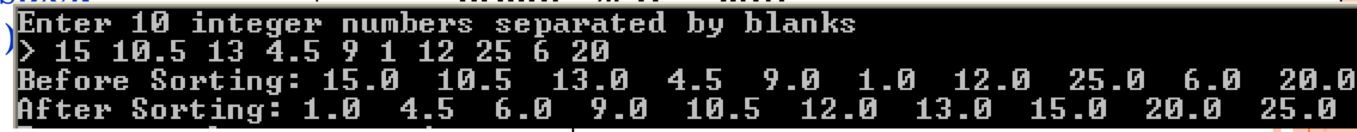
fill is 2. Find the smallest element in subarray list[2] through list[3] and swap it with list[2].

[0]	[1]	[2]	[3]
16	45	74	83

Selection Sort Implementation

```
#include <stdio.h>
#define SIZE  10

void selection_sort(double a[], int size);
void read_array(double a[], int size);
void print_array(double a[], int size);
int find_min(double a[], int start, int size);
void swap(double *a, double *b);

int main(void) {
     double x[SIZE];
     int    i;

     read_array(x, SIZE);
     printf("Before Sorting: ");
     print_array(x, SIZE);
     selection_sort(x, SIZE);
     printf("After Sorting: ");
     print_array(x, SIZE);

    system("pause");
    return 0;
}
void selection_sort(double a[], int size) {
    int i, min_pos;

    for (i = 0; i< size-1; i++) {
       min_pos = find_min(a, i, size);
       swap(&a[i], &a[min_pos])
    }
}
```

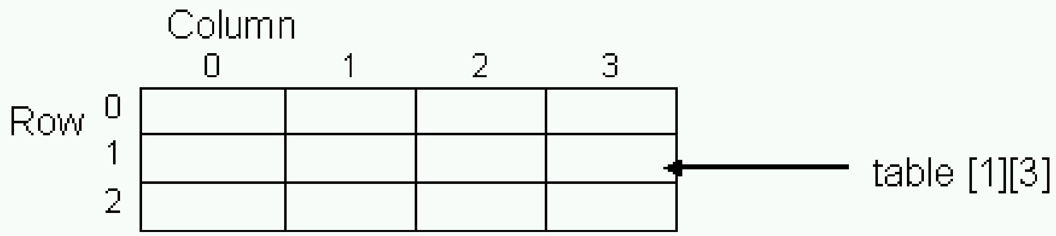
```
int find_min(double a[], int start, int size) {
   int i, min_index = start;

   for (i=start+1; i<size; i++)
      if (a[i] < a[min_index])
        min_index = i;

    return min_index;
}
void swap(double *a, double *b) {
    double temp = *a;
    *a = *b;
    *b = temp;
}
void read_array (double a[], int size) {
    int i;
    printf("Enter %d integer numbers separated by blanks\n>
", size);
    for  (i = 0;  i < size;  ++i)
       scanf("%lf", &a[i]);
}
void print_array(double a[], int size) {
      int i;

      for (i = 0;  i < size;  ++i)
          printf("%.1f  ", a[i]);
```



```
Enter 10 integer numbers separated by blanks
> 15 10.5 13 4.5 9 1 12 25 6 20
Before Sorting: 15.0  10.5  13.0  4.5  9.0  1.0  12.0  25.0  6.0  20.0
After Sorting: 1.0  4.5  6.0  9.0  10.5  12.0  13.0  15.0  20.0  25.0
```

Introduction to 2-D Arrays

- A 2-D array is a contiguous collection of variables of the same type, that may be viewed as a table consisting of rows and columns.



- The same reason that necessitated the use of 1-D arrays can be extended to 2-D and other multi-D Arrays.
- For example, to store the grades of 30 students, in 5 courses require multiple 1-D arrays.
- A 2-D array allows all these grades to be handled using a single variable.
- This idea can be easily extended to other higher dimensions.
- Thus, we shall focus only on 2-D arrays.

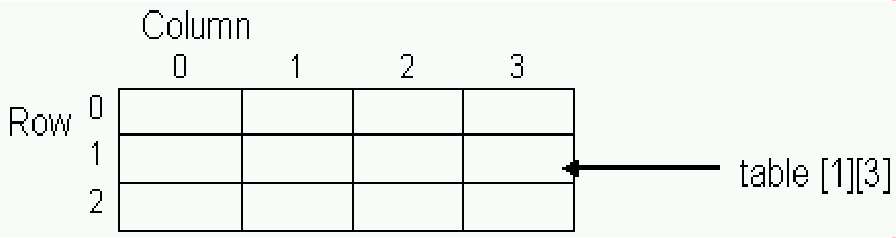
Declaration of 2-D Arrays

- A 2-D array variable is declared by specifying the type of elements, the name of the variable, followed by the number of rows and number of columns – each is a separate bracket:
- The following declares a 2-D array, *table*, having 3 rows and 4 columns.

```
int table[3][4];
```

- Both rows and columns are indexed from zero. So the three rows have indexes 0, 1 and 2 and four the columns have 0, 1, 2, 3.
- As we saw in 1-D array, it is a good practice to declare the sizes as constants. So a better declaration for the above is:

```
#define ROWS 3
#define COLS 4

int table[ROWS][COLS];
```



Accessing 2-D Array elements

- A particular element of a 2-D array, table, is referenced by specifying its row and column indexes:

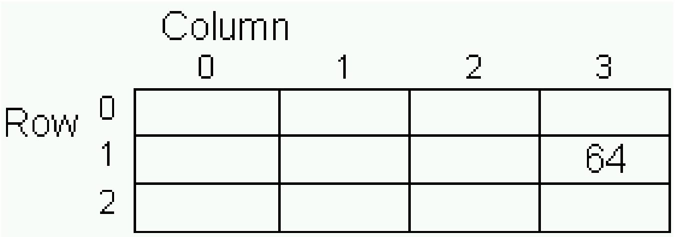
  table[RowIndex][ColumnIndex]

- For example, given the declaration:

  int table[3][4];

- The following stores 64 in the cell with row index 1, column index 3.

  table[1][3] = 64;

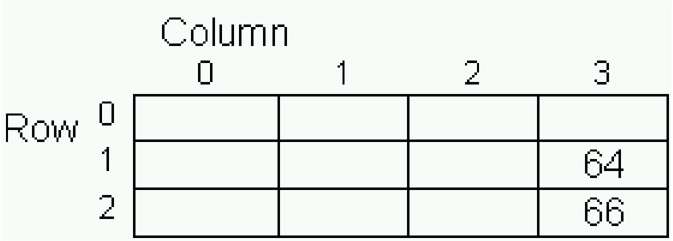


Column

		0	1	2	3
Row	0				
	1				64
	2				

- We use the same format to refer to an element in an expression:

  table[2][3] = table[1][3] + 2;



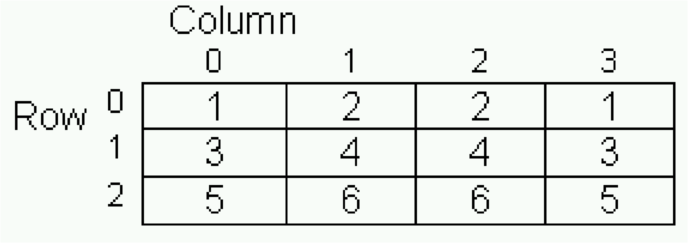
Column

		0	1	2	3
Row	0				
	1				64
	2				66

Initialization of 2-D Arrays

- As with 1-D arrays, if we already have the values to assign to the array at the point of declaration, then we can declare and initialize the 2-D array at the same time.
- The format is similar to 1-D array initialization except that a nested list is used, where each inner list represents a row.
- For example, the following declares and initializes our table.

```
int table[3][4] = { {1,2,2,1}, {3,4,4,3}, {5,6,6,5} }
```



		Column			
		0	1	2	3
Row	0	1	2	2	1
	1	3	4	4	3
	2	5	6	6	5

- Like 1-D array, if you provide less values than the declared size, the remaining cells are set to zero.
- However, unlike 1-D array where you can skip the size, here you must give at least the number of columns.

```
int table[][4] = { {1,2,2,1}, {3,4,4,3}, {5,6,6,5} }     //OK
int table[][] = { {1,2,2,1}, {3,4,4,3}, {5,6,6,5} }      //WRONG!
```

PROCESSING 2-D ARRAYS

- To process 2-D array, we need to extend the loop we normally use with 1-D array to nested loops. This can be done either row-wise or column-wise.
- To process the elements row-wise, we use:

```
for(int rowIndex = 0; rowIndex < ROWS; rowIndex++){
   //process row# rowIndex;
}
```

  - But processing a row involves processing each element of that row; so the complete algorithm is:

```
for(int rowIndex = 0; rowIndex < ROWS; rowIndex++){
   //  process row# rowIndex
   for(int columnIndex = 0; columnIndex < COLUMNS; columnIndex++)
      //process element array[rowIndex][columnIndex]
}
```

- To process elements of a 2-D array column-wise, we use:

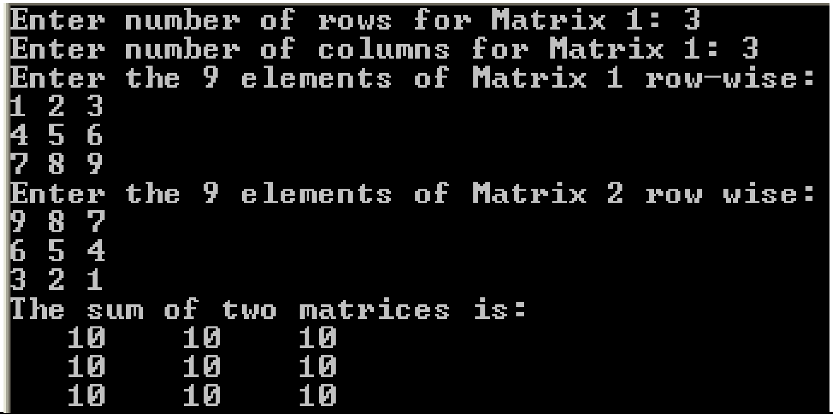
```
for(int columnIndex = 0; columnIndex < COLUMNS; columnIndex++){
   //  process column# columnIndex
   for(rowIndex = 0; rowIndex < ROWS; rowIndex++)
      process element array[rowIndex][columnIndex]
}
```

Example 14

```
/*reads two marices from the user and add them */
#include<stdio.h>
#define ROWS 10
#define COLS 10

int main (void)  {
   int i, j, a[ROWS][COLS], b[ROWS][COLS],  c[ROWS][COLS] = {0},  rows,
    cols;

   printf("Enter number of rows for Matrix 1: ");
   scanf("%d", &rows);
   printf("Enter number of columns for Matrix 1: ");
   scanf("%d", &cols);
   printf("Enter the %d elements of Matrix 1 row-wise: \n", rows * cols);

   for(i=0; i<rows; i++)  {   // reading matrix a
      for(j=0; j<cols; j++)
         scanf("%d", &a[i][j]);
    }

   printf("Enter the %d elements of Matrix 2 row wise: \n", rows * cols);
   for(i=0; i<rows; i++) {    // reading matrix b
      for(j=0; j<cols; j++)
         scanf("%d", &b[i][j]);
   }
```

EXAMPLE 14 …

```
  /* Addition of two matrices */
  for(i=0; i<rows; i++)  {
      for(j=0; j<cols; j++)
          c[i][j]=a[i][j] + b[i][j];
  }

  /*Print sum of two matrices */
  printf("The sum of two matrices is: \n");
  for(i=0; i<rows; i++) {
      for (j=0; j<cols; j++)
           printf("%5d ", c[i][j]);
       printf("\n");
  }

  system("pause");
  return 0;
}
```



```
Enter number of rows for Matrix 1: 3
Enter number of columns for Matrix 1: 3
Enter the 9 elements of Matrix 1 row-wise:
1 2 3
4 5 6
7 8 9
Enter the 9 elements of Matrix 2 row wise:
9 8 7
6 5 4
3 2 1
The sum of two matrices is:
   10    10    10
   10    10    10
   10    10    10
```

2-D Arrays as parameters to functions

- As with 1-D arrays, it is possible to declare functions that take 2-D array as parameter.
- However, one problem here is that in declaring the prototype of the function, we must specify at least the number of columns of the array, thus making the function less flexible.
- One solution to this problem is to use a constant defining the maximum number of columns and use additional parameter to receive the actual size of the array:

```
void print_2d_array(int a[][COLS], int rows, int cols);
```

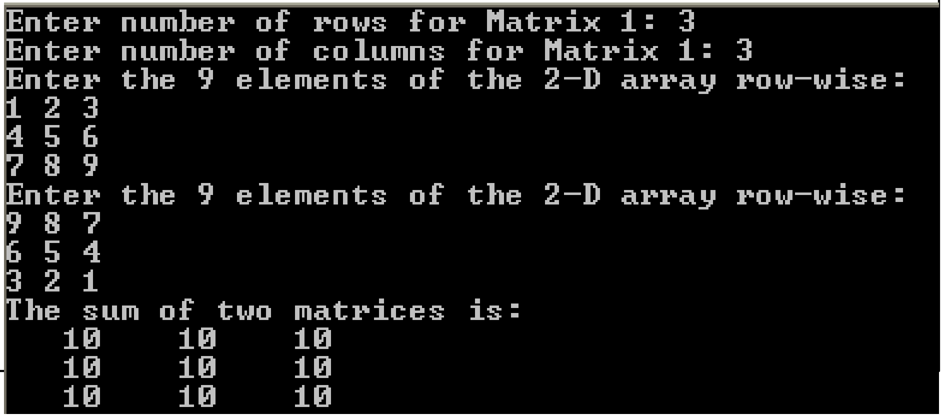
- While this solution makes the function a little more flexible, it is not a perfect solution since the function is not self-contained – it depends on the pre-processor constant COLS.
- Calling functions that take 2-D array as argument is same as calling functions that take 1-D array. Just give the name of the array with no brackets.

EXAMPLE 15

```
#include <stdio.h>
#define ROWS 10
#define COLS 10

void read_2d_array(int [][COLS], int rows, int cols);
void add_2d_arrays(int a[][COLS], int b[][COLS], int c[][COLS], int rows, int cols);
void print_2d_array(int a[][COLS], int rows, int cols);

int main(void) {
    int i, j, a[ROWS][COLS], b[ROWS][COLS],  c[ROWS][COLS];
    int rows, cols;

    printf("Enter number of rows for Matrix 1: ");
    scanf("%d", &rows);
    printf("Enter number of columns for Matrix 1: ");
    scanf("%d", &cols);

    read_2d_array(a, rows, cols);  // reading matrix a
    read_2d_array(b, rows, cols);  // reading matrix a

    add_2d_arrays(a, b, c, rows, cols);    /* Addition of two matrices */

    printf("The sum of two matrices is: \n");
    print_2d_array(c, rows, cols);  /*Print sum of two matrices */

    system("pause");
    return 0;
}
```

Example 15 …

```
void read_2d_array(int a[][COLS], int rows, int cols) {
    int i, j;

   printf("Enter the %d elements of the 2-D array row-wise: \n", rows * cols);

   for(i=0; i<rows; i++) {
       for(j=0; j<cols; j++)
         scanf("%d", &a[i][j]);
   }
}
void add_2d_arrays(int a[][COLS], int b[][COLS], int c[][COLS], int rows, int cols) {
    int i, j;

    for (i=0; i<rows; i++) {
       for (j=0; j<cols; j++)
         c[i][j] = a[i][j] + b[i][j];
    }
}
void print_2d_array(int a[][COLS], int rows, int cols) {
   int i, j;

   for(i=0; i<rows; i++) {
      for (j=0; j<cols; j++)
          printf("%5d ", a[i][j]);
       printf("\n");
   }
}
```



```
Enter number of rows for Matrix 1: 3
Enter number of columns for Matrix 1: 3
Enter the 9 elements of the 2-D array row-wise:
1 2 3
4 5 6
7 8 9
Enter the 9 elements of the 2-D array row-wise:
9 8 7
6 5 4
3 2 1
The sum of two matrices is:
   10    10    10
   10    10    10
   10    10    10
```