```
    for ( i = 0; i < 3; i++ )
        printf ( "\t%d", sum[i] );
    printf ( "\n" );
}
```

  - Passing 2-dimensional arrays to functions
    * A more tricky problem than passing one-dimensional arrays
    * In the function header and the prototype, you need to declare the second dimension
    * Passing two-dimensional array grades
      · Function header

```
int min_grade ( int grades[][10] ) { ... }
```

      · Prototype

```
int min_grade ( int [][10] );
```

      · Function call

```
min = min_grade ( grades );
```

- Multi-dimensional arrays (higher than two dimensions)

Character arrays (strings)

- Special arrays to represent strings
- The string is always terminated by a zero (the NULL character, or '\0'
- Consider the following segment

```
char name[4];
main()
{
name[0] = 'S';
name[1] = 'a';
name[2] = 'm';
name[3] = '\0';
}
```

- Strings are of variable length
  - Even if the array name had been declared with 50 elements, the string terminator '\0' makes the string behave like an array of three characters

- A table can be represented by using a two-dimensional array
- Two-dimensional array
  - Consists of rows and columns
  - Each element is identified by two subscripts
    - First subscript identifies the row index
    - Second subscript identifies the column index
  - Example of a 2×3 array

77	68	86
73	96	87
89	78	70

  - The element in the ith row and jth column in array a is referred to by `a[i][j]`
  - A double subscripted array element cannot be referred to as `a[i,j]`
  - Initializing the 2-dimensional array

    ```
    int b[2][3] = { { 77, 68, 86 }, { 73, 96, 87 }, { 89, 78, 70 } };
    ```

  - Printing the 2-dimensional array (and using the elements)

    ```
    #include <stdio.h>

    main()
    {
        int b[3][3] = { { 77, 68, 86 }, { 73, 96, 87 }, { 89, 78, 70 } };
        int i, j;

        for ( i = 0; i < 3; i++ )
        {
            for ( j = 0; j < 3; j++ )
                printf ( "\t%d", b[i][j] );
            printf ( "\n" );
        }
    }
    ```

  - Another way to initialize the two-dimensional arrays

    ```
    int b[][3] = { { 77, 68, 86 }, { 73, 96, 87 }, { 89, 78, 70 } };
    ```

    - The first array subscript can be omitted but the second is required
  - Taking the sum of each column in the array

    ```
    #include <stdio.h>

    main()
    {
        int b[][3] = { { 77, 68, 86 }, { 73, 96, 87 }, { 89, 78, 70 } };
        int sum[3] = { 0, 0, 0 };
        int i, j;

        for ( i = 0; i < 3; i++ )
            for ( j = 0; j < 3; j++ )
                sum[j] += b[i][j];
    ```

  - Pick up the middle element (a[n/2])

Searching arrays

- Search for an element (known as the key) in the array
- Linear search
  - Examine each element in the array and return the index of the element that matches

```
/*****************************************************************************/
/* main.c: Main driver for the linear search program                         */
/*****************************************************************************/
#include <stdio.h>

#define ASIZE 7
int lin_search ( int [], int, int );

main()
{
    int a[] = {3, 1, 5, 2, 9, 4, 6};
    int key, index;

    printf ( "Enter an integer to find its index: " );
    scanf ( "%d", &key );

    index = lin_search ( a, ASIZE, key );

    if ( index == -1 )
        printf ( "Sorry, the item does not exist in the array\n" );
    else
        printf ( "The item exists in location %d in the array\n", index );
}
/*****************************************************************************/
/* lin_search.c: Linear search in an array to return the index of key        */
/*****************************************************************************/
int lin_search ( const int a[], const int n, const int key )
{
    int i;                /* Counters              */

    for ( i = 0; i < n; i++ )
        if ( a[i] == key )
            break;

    return ( i == n ? -1 : i );
}
```

Arrays with multiple subscripts

- Higher dimensional arrays are common in programming
- C is capable of supporting subscripts up to 12 levels deep

Sorting arrays

- Sorting a ten element array in ascending order using *bubble sort*

```
/*****************************************************************************/
/* main.c: Main driver for the bubble sort program                           */
/*****************************************************************************/

#include <stdio.h>

#define ASIZE 7

void bubble_sort ( int [], int );

main()
{
    int a[] = {3, 1, 5, 2, 9, 4, 6};
    int i;

    bubble_sort ( a, ASIZE);

    for ( i = 0; i < ASIZE; i++ )
        printf ( "%d  ", a[i] );

    printf ( "\n" );
}
/*****************************************************************************/
/* bubble.c: Bubble sort function                                            */
/*****************************************************************************/

void bubble_sort ( int a[], const int n )
{
    int i, pass;               /* Counters              */
    int tmp;

    for ( pass = 1; pass < n; pass++ )
        for ( i = 0; i < n - 1; i++ )
            if ( a[i] > a[i+1] )
            {
                tmp = a[i];
                a[i] = a[i+1];
                a[i+1] = tmp;
            }
}
```

Computing mean, median, and mode

- You already have a function to compute mean (`average.c`)
- Computing median
  - Sort the array by calling `bubble_sort`

  - You should be careful when initializing something to other than zero
  - The initialization string should not have more elements than the array can hold
  - You could also omit the size of the array when it is being initialized

    ```
    int grade[] = { 58, 77, 23, 85, 30, 19, 96, 48, 41, 18 };
    ```

    achieves the same effect as before

- The size of the array can be (and should be) declared as a constant by using a `#define` directive
  - By doing so, you can change the size in the directive and it is reflected everywhere you use the constant
  - Do not use a semicolon at the end of the define directive
  - The size of the array cannot be declared as a variable
  - It is not possible to refer to an element outside the array bounds
    * If the array limit is 10 elements, you cannot refer to element 12
    * Doing so will give you strange results, and may even etrminate the program with a core dump
    * Array subscript cannot be negative
- Functions that contain automatic arrays (local to the function) should have the arrays declared as `static` so that it is not created each time the function is called (or the block is entered)
- It is always a good idea to initialize the elements in the array before use

Passing arrays to functions

- In the function call, the name of the array is to be specified without the square brackets
- Example
  - Use the previous declaration for `grade` to be passed to the function `int avg_grade`

    ```
    printf ( "%d", avg_grade ( grade, NUM_STUDENTS ) );
    ```

  - The function `avg_grade` can be written as

```
int avg_grade ( grade[], n )
{
    int sum, i;

    for ( i = 0, sum = 0; i < n, i++ )
        sum += grade[i];

    return ( sum / n );
}
```

- Arrays can be used to simulate call by reference in C
  - If the contents of an array are modified within the function, the change is reflected in the calling function as well
- Special type qualifier `const`
  - Prevents modification of the variable (or array elements)
  - Makes the qualified variable (or array) constant in the function body
  - Any attempt to modify a `const` results in a compile time error

- Refer to the precedence chart to find the order of evaluation of array subscripts with respect to other operators

Declaring arrays

- You reserve space in memory for the number of elements by specifying the type of elements and the actual number of elements you need
- The array grade used above is defined by

```
int grade[10];
```

  - Notice that we asked for 10 elements but they will be subscripted from 0 to 9

Examples using arrays

- Finding the average score of all the students on a test

```
/*****************************************************************************/
/* score_mean : Program to compute the average score for ten students        */
/*****************************************************************************/

#include <stdio.h>

main()
{
    char line[100];          /* Input buffer              */
    int grade[10];           /* Ten students in the class */
    int i;                   /* Counter                   */
    int sum, avg;            /* Total and average         */

    for ( i = 0; i < 10; i++ )
    {
        printf ( "Please enter the grade for student %d : ", i );
        fgets ( line, sizeof ( line ), stdin );
sscanf ( line, "%d", &grade[i] );
    }

    sum = avg = 0;
    for ( i = 0; i < 10 ; sum += grade[i++] );
    avg = sum / 10;

    printf ( "The average score is: %d\n", avg );
    exit ( 0 );
}
```

- Initializing arrays
  - An array can be initialized in the definition stage by using curly braces to enclose the elements, with the elements being separated by commas
  - The array grade could be defined as

```
int grade[10] = { 58, 77, 23, 85, 30, 19, 96, 48, 41, 18 };
```

Arrays

What are arrays?

- A group of memory locations that all have the same type and share the same name
- A particular element in an array can be referenced by the common name and the location number within that name
- Example – Grades awarded to students in the test

`student[0]`	Madonna Cicconne
`student[1]`	Cindy Crawford
`student[2]`	Jane Doe
`student[3]`	John Doe
`student[4]`	Mary Doe
`student[5]`	John Henry
`student[6]`	Kathy Ireland
`student[7]`	Jane Smith
`student[8]`	John Smith
`student[9]`	Mary Smith

`grade[0]`	58
`grade[1]`	77
`grade[2]`	23
`grade[3]`	85
`grade[4]`	30
`grade[5]`	19
`grade[6]`	96
`grade[7]`	48
`grade[8]`	41
`grade[9]`	18

- As normal with C, the first element in the array is the zeroth element
- The array `grade` contains ten elements, numbered from 0 to 9
- The name of the array is `grade`
- The elements are referred to as `grade[0]`, `grade[1]`, and so on
- You can also use an expression to refer to the elements, for example, `grade[a+b]` refers to `grade[5]` if `a` is 2 and `b` is 3
- The value contained in `grade[0]` is 58
- You can print the value in a single element, or the sum of elements 3 and 7 as
```
printf ( "%d", grade[5] );
printf ( "%d", grade[3] + grade[7] );
```
- Off-by-one error
  - Difference between element 1 and the first element
- The square brackets used to specify the subscript of an array form an operator in C (This is important to remember and we'll come back to it later)