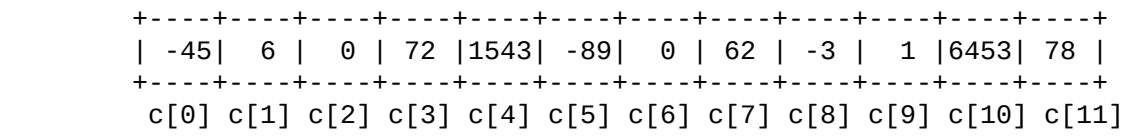
Chapter 6: Arrays

* Arrays are data structures consisting of related data items of the same type.
* Arrays are "static" entities in that they remain the same size throughout program execution (they may be of automatic storage class).

Arrays

* An array is a group of memory locations related by the fact that they all have the same name and the same type.
* To refer to a particular location or element in the array, we specify:
    1. the name of the array
    2. the position number of the particular element
* E.g.



* The first element in every array is the zeroth element; i.e., "c[0]".
* In general, the ith element of array "c" is referred to as "c[i-1]".
* The position number (must be an integer or an integer expression) contained within square brackets is more formally called a subscript, or index.
* E.g. a=5, b=6

```
c[a+b] += 2;
```

adds 2 to array element c[11].

Declaring Arrays

* The programmer specifies the types of each element and the number of elements required by each array.
* The computer reserve the appropriate amount of memory.
* E.g.

```
int c[12];
```

is used to tell the computer to reserve 12 elements for integer array "c".

* E.g.

```
int b[100], x[27];
char str[50];
float reading[200];
```

* E.g.

```
#include <stdio.h>
main()
{
    int n[10], i;

    for (i=0; i<=9; i++)
        n[i] = 0;
    printf("%s%13s\n", "Element", "Value");
    for (i=0; i<10; i++)
        printf("%7d%13d\n", i, n[i]);
    return 0;
}
```

* The elements of an array can also be initialized in the array declaration by following the declaration with an equals sign and a comma-separated list (enclosed in braces) of initializers.
* E.g.

```
#include <stdio.h>
main()
{
    int i, n[10] = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

    printf("%s%13s\n", "Element", "Value");
    for (i=0; i<10; i++)
        printf("%7d%13d\n", i, n[i]);
    return 0;
}
```

* If there are fewer initializers than elements in the array, the remaining elements are automatically initialized to zero.
* It is important to remember that arrays are not automatically initialized to zero, just like other variables.
* If the array size is omitted from a declaration with an initializer list, the number of elements in the array will be the number of elements in the initializer list.
* E.g. five-element array

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

* E.g.

```
#include <stdio.h>
#define SIZE 10
main()
{
    int s[SIZE], j;

    for (j=0; j <= SIZE - 1; j++)
        s[j] = 2 + 2 * j;
    printf("%s%13s\n", "Element", "Value");
    for (j=0; j<SIZE; j++)
        printf("%7d%13d\n", j, s[j]);
    return 0;
}
```

* The line "#define SIZE 10" defines a symbolic constant "SIZE" whose represents "10".
* A symbolic constant is an identifier that is replaced with replacement text by the C preprocessor before the program is compiled.
* Using symbolic constants to specify array size makes programs more scalable.

* E.g.

```
#include <stdio.h>
#define SIZE 12
main()
{
    int a[SIZE] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45},
        i, total = 0;
    for (i=0; i<SIZE; i++)
        total += a[i];
    printf("Total of array element values is %d\n", total);
    return 0;
```

```
        }
```

* A string such as "hello" is really an array of individual characters in C.
* E.g.

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

* It is important to note that the string "first" contains five characters plus a special string termination character called the null character ('\0').
* Thus, array "string1" actually contains six elements.
* All strings in C end with the null character.
* A character array representing a string should always be declared large enough to hold the number of characters in the string and the terminating null character.
* E.g. string1[0] is the character 'f' and string1[3] is the character 's'.

* We also can input a string directly into a character array from the keyboard using "scanf" and the conversion specification "%s".
* E.g.

```
        char string2[20];
        scanf("%s", string2);
```

* The name of the array is passed to "scanf" without the preceding "&" used with other variables.
* "scanf" can write beyond the end of the array.
* A character array representing a string can be output with "printf" and the "%s" conversion specifier.
* E.g.

```
        printf("%s\n",string2);
```

* The characters of the string are printed until a terminating null character is encountered.
* E.g.

```
        #include <stdio.h>
        main()
        {
            char string1[20], string2[] = "string literal";
            int i;

            printf("Enter a string: ");
            scanf("%s", string1);
            printf("string1 is: %s\nstring2 is: %s\n"
                    "string1 with spaces between characters is:\n",
                    string1, string2);
            for (i=0; string1[i] != '\0'; i++)
                printf("%c ", string1[i]);
            printf("\n");
            return 0;
        }
```

* In functions that contain automatic arrays where the function is in and out of scope frequently, make the array "static" so it is not created each time the function is called.

Passing Arrays to Functions

* To pass an array argument to a function, specify the name of the array without any brackets.
* E.g.

```
        int hourlyTemp[24];
        ....
```

```
        modifyArray(hourlyTemp,24);
        ....
```
* When passing an array to function, the array size is often passed so the function can process the specific number of elements in the array.
* The name of the array is actually the address of the first element of the array.
* Because the starting address of the array is passed, the called function knows precisely where the array is stored.
* Therefore, when the called function modifies array elements in its function body, it is modifying the actual elements of the array in their original memory locations.
* E.g.

```
        #include <stdio.h>
        main()
        {
            char array[5];
            printf("array = %p\n&array[0] = %p\n", array, &array[0]);
            return 0;
        }
```
* Although entire arrays are passed simulated call by address, individual array elements can be passed call by values exactly as simple variables are.
* E.g.

```
        #include <stdio.h>
        main()
        {
            char str[]="hello";
            foobar(str[3]);
            ....
        }
```

* For a function to receive an array through a function call, the function's parameter list must specify that an array will be received.
* E.g.

```
        void modifyArray(int b[], int size)
        {
            int j;
            for (j=0; j<size; j++)
                b[j] *= 2;
        }
```
* The size of the array is not required between the array brackets. If it is included, the compiler will ignore it.

* When an array parameter is preceded by the "const" qualifier, the elements of the array become constant in the function body, and any attempt to modify an element of the array in the function body results in a compiler time error.
* E.g.

```
        #include <stdio.h>
        void tryToModifyArray(const int []);
        main()
        {
            int a[] = {10, 20, 30};
            tryToModifyArray(a);
            printf("%d %d %d\n", a[0], a[1], a[2]);
            return 0;
        }
```

```
        void tryToModifiyArray(const int b[])
        {
            b[0] /= 2  /* error */
            b[1] /= 2  /* error */
            b[2] /= 2  /* error */
        }
```

Multiple-Subscripted Arrays
---

* A common use of multiple-subscripted arrays is to represent tables of values (in rows and columns).
* Tables or arrays that require two subscripts (indexes) to identify a particular element are called double-subscripted (two-dimensional) arrays.

* Every element in array "a" is identified by a name of form "a[i][j]": "a" is the name of the array, and "i" and "j" are the indexes that uniquely identify each element in "a".
* A multiple-dimensional array can be initialized:

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

* E.g.

```
        #include <stdio.h>
        void printArray(int [][3]);
        main()
        {
            int array1[2][3] = {{1,2,3},{4,5,6}},
                array2[2][3] = {1,2,3,4,5},
                array3[2][3] = {{1,2},{4}};

            printf("Values in array1 by row are:\n");
            printArray(array1);

            printf("Values in array2 by row are:\n");
            printArray(array2);

            printf("Values in array3 by row are:\n");
            printArray(array3);

            return 0;
        }

        void printArray(int a[][3])
        {
            int i,j;
            for (i=0;i<2;i++)
            {
                for (j=0;j<3;j++)
                    printf("%d ", a[i][j]);
                printf("\n");
            }
        }
```

* Execrise

Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10. Place the forty responses in an integer array and summarize the result of the poll (by values and histogram).