Arrays

1. Arrays

The array is another data type for storing values. Unlike a regular single variable where it can store only one value at a time, an array can store many values in different locations using the same name. An index is used to access the different locations in an array. An array of size 10 has index locations from 0 to 9.

To declare an integer array called *n* having 10 locations:

```
int n[10];
```

Declaring a string array called *day* and initializing it with the seven names:

```
string day[7] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
```

To assign the number 13 into location 2 of the array:

```
n[2] = 13;
```

```
int main () {

    int n[10];    // declare an array called n of size 10
    int m[5] = {8, 5, 4, 3, 9};     // initialize array m with five numbers
    int i;

    n[0] = m[1] + 2;    // accessing values in the array
    cout << n[0];       // the number 7 will be printed out

//input 10 values into the 10 locations of the array n
    for(i=0; i<10; i=i+1){
      cin >> n[i];
    }

//print the 10 values from the 10 locations of the array
    for(i=0; i<10; i=i+1){
      cout << n[i];
    }

    return 0;
}
```

2. 2-dimensional Arrays

To declare a two-dimension array of floating point numbers called *table* having 3 rows and 2 columns:

```
float table[3][2];
```

To assign the number 3.1415 into row 2, column 0 of the array:

```
table [2][0] = 3.1415;
```

	Column 0	Column 1
Row 0		
Row 1		
Row 2	3.1415	

To declare and initialize a 2-dimensional array having 2 rows and 5 columns:

```
int n[2][5] = {{2,4,6,8,10},{1,3,5,7,9}};
```

To access row 1 column 3 of the array:

```
cout << n[1][3];    // what will be printed out?
```

or

```
if (n[1][3] > 5) …
```

3. Parallel Arrays

By using the same subscript for two or more arrays, you can build relationships between data stored in these arrays. This is especially useful when the related data is of unlike types. For example, in a payroll system where you want to store the names of employees and the hours worked by each employee, you will use one array of type string to store the names and a second array of type int to store the hours worked. Using the same index for both arrays will allow you to access the name and hours worked for that one employee. Notice that this is similar to a 2-dimensional array in the sense that the two separate arrays are like two columns in a 2-dimensional array. The difference, however, is that the two parallel arrays can be of different data types, whereas for a 2-dimensional array, all columns have the same type.

```
#include <iostream>
#include <iomanip>  // needed for setw
using namespace std;

int main() {
  const int SIZE = 5;
  string names[SIZE];
  unsigned int hours[SIZE];

  for (int i=0; i<SIZE; i++) {
    cout << "Enter the name and hours worked for employee " << i+1 << "? ";
    cin >> names[i] >> hours[i];
  }

  cout << left << setw(10) << "Name" << right << setw(10) << "Hour" << endl;
  for (int i=0; i<SIZE; i++) {
    cout << left << setw(10) << names[i] << right << setw(10) << hours[i] <<
endl;
  }
}
```

Subscript/Row
0
1
2
3
4

names
John
Jane

hours
3
7

4. *Exercises* (Problems with an asterisk are more difficult)

1. Write a program using a loop to input 10 numbers. After you have entered these 10 numbers, then the program will print out these 10 numbers.

2. Write a program with an array of size 10. Initialize the array with numbers from 1 to 50 of your choice. Now calculate and print out the sum of these numbers.

3. Write a program with an array of size 10. Initialize the array with numbers from 1 to 50 of your choice. Now calculate and print out the average of these numbers.

4. Write a program to generate 10 random numbers from 1 to 50. Store these 10 numbers in an array. Now find and print out the largest of these numbers.

5. Write a program to generate 10 random numbers from 1 to 50. Store these 10 numbers in an array. Now find and print out the largest and second largest of these numbers.

6. Write a program to enter an integer number and then generate the multiplication table from 1 to 10 for that one number. Put these numbers into an array of size 10. After that, print out the table from the array.

7. Write a program with an array of size 10. Initialize the array with numbers of your choice. Have the user enter a number, and then search for this number in the array. Print out the message "number found" if the number entered is found in the array.

8. * Repeat question 7 but also print out the message "number NOT found" if the number is not found in the array.

9. Write a program to print out a message depending on an input number. If the input number is 1 then print out the message "Good morning", if the number is 2 then print out the message "Good afternoon", if the number is 3 then print out the message "Good evening", if the number is 4 then print out the message "Good night" and if the number is anything else then print out the message "Invalid input." The program repeats if the input number is invalid.

    a) Use nested if statements to do this.

    b) Use a switch statement to do this.

    c) Use parallel arrays to do this.

10. * Create a two-dimensional 10×10 array and have the program automatically fill it with the numbers for the 10×10 multiplication table. You need to use two for loops for this. To check that the program is correct you will need to print out this two-dimensional array.

11. ** Sorting. Write a program with an array of size 10. Initialize the array with numbers of your choice but not in any order. Now print these numbers in ascending order.

12. ** Write a program to generate 11 random numbers from 1 to 50. Store these 11 numbers in an array. Now calculate and print out the median of these numbers.

    Reference: https://en.wikipedia.org/wiki/Median

13. Write a program to input an integer. Print out whether the number entered is a prime number or not.

14. *** Use the Sieve of Eratosthenes algorithm for finding all the prime numbers from 2 to 500. The algorithm:

    1. Create a list of consecutive integers from two to n: (2, 3, 4, ..., n).
    2. Initially, let p equal 2, the first prime number.
    3. Cross out from the list all multiples of p greater than p.
    4. Find the first number remaining in the list greater than p (this number is the next prime); let p equal this number.
    5. Repeat steps 3 and 4 until p^2 is greater than n.
    6. All of the remaining numbers in the list are prime.

    Hint: For step 1, you will need to use an array to keep track of whether a number has been crossed out or not. The indexes for the array are the integers. Initialize the array with 0's. A 0 in a location means that that number is not crossed out, and a 1 in that location means that that number has been crossed out. For step 3, you can cross out a number simply by changing the content of that number index location to a 1.

    Reference: http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

