Chapter 5 – Arrays and Strings

Up until now, you haven't seen much difference between Java, Objective-C, and C++. The syntax has been pretty close and the big differences have been with the libraries. It's no surprise that Java adopted the C syntax; it made it easier for programmers to move from one language to the other. That's the reason why it is important to learn the basics and concepts of one language really well; it carries over to other languages. As we move on to more complicated things, there will be larger differences between these languages especially in the area of how the standard libraries are put together.

Once again, we will backtrack to solidify our knowledge. Before using variables, they must be declared. We can use either of the 2 syntaxes:

```
dataType identifier;
dataType identifier = initialValue;
```

It's always safer to use an initial value. The compiler will warn you if you try to use a variable without assigning it a value. Remember that the value will be unknown if you don't assign it something. It is also important to declare the variable before using it (higher up in the source code).

When we use *arrays* with control statements, our code becomes easier to manage. It is easier to initialize and iterate. To declare basic arrays, we use the following syntaxes:

```
dataType identifier[size];
dataType identifier[sizeN] = {element1, element2, … elementN};
dataType identifier[] = {element1, element2, … elementN}; // automatically size of N
```

To access any array element, we use:

```
identifier[elementNumber]    // can be used to access or assign
```

If we look back to Example 3.2 (min and max of a list), you can see that a lot of code is required if we want to find the minimum and maximum of a list of 10 numbers. We would have to specifically access num1, num2, num3, … However with arrays, we can use num[0], num[1], num[2], …, but we can also access with a variable, num[i]. Arrays always start with index 0 and end with *listSize-1*; this is a major source of bugs when using arrays: accessing element *listSize* will produce an array out of bounds error.

> **Exercise 5.1: Modify the template Example_5_1 to calculate the maximum, sum, and average of the list of 10 numbers. Recall that average=sum/(list size). Sample printout:**

```
The list of numbers:
10 7 -1 23 -8 17 39 2 29 -14
There are 10 numbers in the list

The minimum is: -14
The maximum is: 39
The sum is: 104
The average is: 10.400000    // average should be a double to get the decimals
```

As programmers, we may not be experts in the area that we are doing work. But we still need to be able to do the code and test it. Sometimes after doing the work, we may gain some expertise in the area.

> **Exercise 5.2: Modify the template Example_5_2 to calculate the standard deviation of a list of 20 random integers. Use the non-random list first to make sure you have the right calculations before converting it to the random list. Below is the formal definition and a sample printout:**

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

```
The list of numbers:
10 7 -1 23 -8 17 39 2 29 -14 11 -5 19 8 23 11 14 3 -4 21
There are 20 numbers in the list

The sum is: 205
The mean is: 10.250000
The sum of squared deviations is: 3375.750000
The variance is: 168.787500
The standard deviation is: 12.991824
```

A special type of array is a *string*. It is special because we use it to print out text and information. There aren't many programs that don't use strings. Because of this there are many library methods that work on strings that aren't included with arrays. However, we will start with old school so that you can fully understand how a string library is constructed. There are 2 ways to construct strings using character arrays. One way is to create a character array of a certain length and fill it with characters; this requires meticulous counting. Another way is to create a character array of a maximum length, then fill it with characters and ending the string with a zero byte; this is called a null-terminated string. The problem with this string is that you can overrun the buffer; that is, put in more characters than was allocated causing "memory stomps". So, let's look at Example_5_3.

Exercise 5.3: Modify the template Example_5_3 to perform the following string methods with the specified API (Application Program Interface):

```
void printLeftString(char string[], int len);
       // prints the first len characters of a string
void printRightString(char string[], int len);
       // prints the last len characters of a string
int indexOfString(char string[], char search);
       // returns the array position of search character
void printReverseString(char string[]);
       // prints the string in reverse characters
void printReplaceCharString(char string[], char search, char replace);
       // prints the string replacing a characters
```

It is okay to hard-code this assignment. Sample printout:

```
cstring1 = this is a C style string; length=24
printLeftString(cstring1, 10) = this is a
printRightString(cstring1, 10) = yle string
indexOfString(cstring1, 'a') = 8
printReverseString(cstring1) = gnirts elyts C a si siht
printReplaceCharString(cstring1, 's', '$') = thi$ i$ a C $tyle $tring
```

Exercise 5.4: Modify the string input loop (Example_5_4) and perform a palindrome check on the string. Make the palindrome check a method; you need to come up with your own API. Hint: you will need to include and use *stringLength*. Sample printout:

```
Input a string (q to quit):  palindrome
palindrome is not a palindrome.
Input a string (q to quit):  ABBA
ABBA is a palindrome.
Input a string (q to quit):  mom
```

```
mom is a palindrome.
```

Exercise 5.5: When writing tools, it is important to parse strings for parameters or to do other processing. So in this exercise you must modify indexOfString to:

```
int indexOfStringStart(char string[], char search, int start);
```

Make sure you take the time to properly construct your loops. The italic print is optional, but it may help you to debug the program. You decide which template is appropriate. There are multiple ways of solving this problem. Bonus if you can get it to work with double spaces.

Sample printout:

```
Enter a string (Q to quit): This is a sentence.
This is your string: This is a sentence.
indexOfStringStart(cstring, ' ', 0) = 4
indexOfStringStart(cstring, ' ', 5) = 7
indexOfStringStart(cstring, ' ', 8) = 9
indexOfStringStart(cstring, ' ', 10) = -1
Number are 4 words.

Enter a string (Q to quit): This is  a  sentence.
This is your string: This is  a  sentence.
indexOfStringStart(cstring, ' ', 0) = 4
indexOfStringStart(cstring, ' ', 5) = 7
indexOfStringStart(cstring, ' ', 8) = 8
indexOfStringStart(cstring, ' ', 9) = 10
indexOfStringStart(cstring, ' ', 11) = 11
indexOfStringStart(cstring, ' ', 12) = -1
Number are 4 words.
```

Exercise 5.6: This exercise requires you to convert all lower case letters in a string to upper case. Recall that a char holds an ASCII code, so 'A' is number 65 and 'a' is number 97. You can perform a conditional such as "ch >= 'A'". Sample printout:

```
Enter a string (Q to quit): This is a sentence.
This is your string: This is a sentence.
String in upper case: THIS IS A SENTENCE.
Enter a string (Q to quit): quit
```

Exercise 5.7: We have done exercises to split strings, now it is time to build strings. Use the following API to build a new string:

```
void concatStrings(char string1[], char string2[], char newString[]);
```

Hint: you will need to allocate 3 char buffers and don't forget to add a null terminator.

Sample printout:

```
Enter the first string (Q to quit): This is the first.
Enter the second string: This is the second.
This is string1: This is the first.
This is string2: This is the second.
This is the concat string: This is the first. This is the second.
```

Converting strings into numbers is an important concept because all number entry is done in text and converted to integers or floats. Example_5_5 demonstrates how to convert a decimal valued string into an integer.

Exercise 5.8: Convert a string that contains a binary number into an integer using a method; you can assume that the input is correct. <u>Bonus if you can convert a string containing a hexadecimal number into an integer.</u> Sample printout:

```
Enter a binary value (Q to quit): 1011011
This is your string: 1011011
The decimal value is: 91
Enter a binary value (Q to quit): 1100110011
This is your string: 1100110011
The decimal value is: 819
```

There are many applications for arrays. Just think about any time you make a list. This next exercise is to simulate a deck of cards. We number all the cards from 0 to 51. We will have all the spades from 0 to 12, hearts from 13 to 25, diamonds from 26 to 38, and clubs from 39 to 51. We can get the suit by doing taking the quotient of the card number divided by 13, so spades are 0, hearts are 1, diamonds are 2, and clubs are 3. The rank of the card can be determined by remainder of the card number divided by 13; the syntax is number % 13. I leave it up to you when 0 is "Ace" or "2", then 1 is "2" or "3", … and 12 is "K" or "Ace".

Exercise 5.9: Create 2 arrays, one for suit and one for rank. Ask the user for a card number 0-51, then print out the suit and rank of that card. Sample printout:

```
Enter card number (Q to quit): 0
Ace of Spades
Enter card number (Q to quit): 25
King of Hearts
```

Exercise 5.10: Here is another deck of cards. You will need to determine what arrays and what equations are necessary. Hint: Start with the arrays, then do the equations.

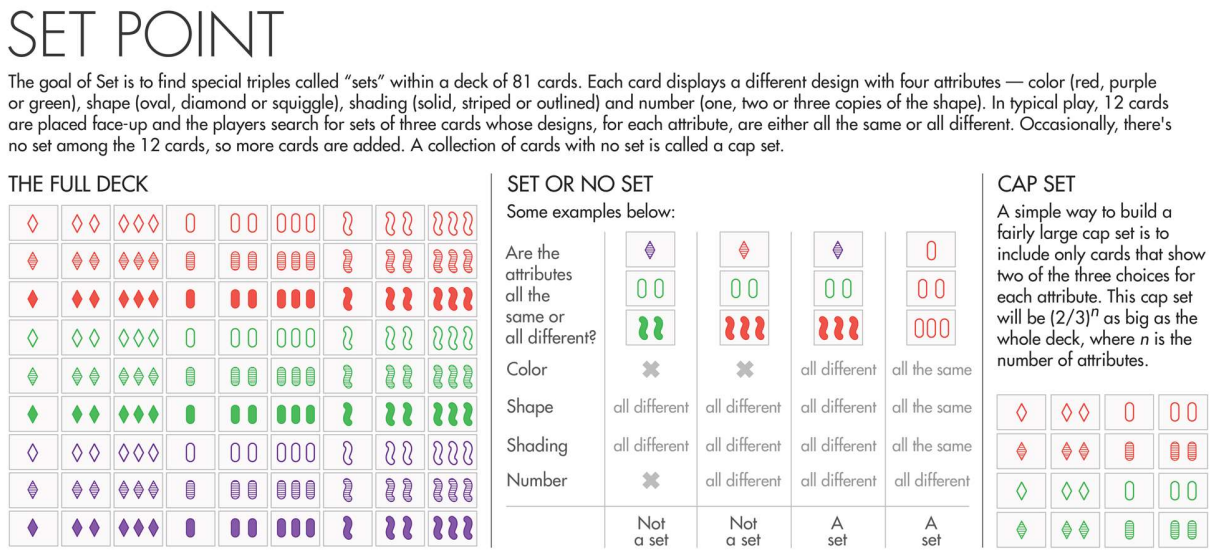

Image Source:

https://d2r55xnwy6nx47.cloudfront.net/uploads/2016/05/SETPoint_2000.png