Using Arrays in C

As programs get larger, the amount of information that must be controlled or manipulated usually also increases. Often this information is of a similar **type** and for such situations there is a format of handling this data that makes the job of the programmer much simpler.

For example, suppose you were asked to make a program which could input and store the average mark (as an integer) for each of the students in the class. With this data you would then like to be able to identify the best mark and the worst lowest mark in the class, and calculate the mean of all of the marks. This would not be difficult if there were only 2 or 3 students in the class, but if there are 20 to 30, it would be a somewhat tedious process.

One way in which we have already used arrays was for storing *strings* or words in our programs. The square brackets [] were used to indicate the number of letters (or **char** type variables) in the word. In much the same way, an array can be created of any data type in a C program, including data types of your own creation.

Key Points for Arrays

- An array is like a **list** that is used to declare a group of variables.
- The number of **elements** in the array is identified using square brackets.
- The first element in a C array has an **index** of zero, shown by [0].
- One of the most common errors in use of arrays is going 'beyond the end' of the array.
- Any arrays to be used in your programs must be declared with all of the other variables.

Examples

An array for a student's 8 course marks: `float myMarks[8];`

An array for letter grades: `char letterGrades[6] = {'A','B','C','D','F'}`

Note: Arrays can be **multidimensional**, for example 2x2 or 3x3, which has some very useful properties in some programs. (In reality, however, these are arrays of arrays, but it accomplishes the same thing)

To show how the use of arrays can make this task MUCH easier, a sample program is given, which you are to type in as a C program and run. Those of you who are skeptical of arrays may consider what the equivalent program would look like without them!

```
/***
An introduction to use of arrays in programming
***/
#include <stdio.h>
int main(void)
{
	int	marks[30],			//This will allow storage of up to 30 marks
		numStudents;			//We will allow the user to enter the number

	int	low=100, high=0,		//This will be used to score high/low scores
		total=0,			//The total of all scores will be used for the mean
		i;
	float	mean = 0;

	printf("\nThe Mark Manipulator!!\n\n");
	printf("How many students are in the class (up to 30) è");
	scanf("%i",&numStudents);

	if(numStudents > 0 && numStudents < 31)			//An OK # of students
	{
		for(i=0; i<numStudents;i++)			//Enter the marks
		{
			//The first entry in a C array goes in spot 0!!!
			printf("\nEnter the mark for student %i è", i+1);
			scanf("%i",&marks[i]);

			if(marks[i]>high)		//Is this a new 'best' mark??
			{
				high = marks[i];	//If so, save it as the high
			}
			if(marks[i] < low)		//Is this a new 'low' mark?
			{
				low = marks[i];		//If so, save it as the low
			}
			total += marks[i];		//"ACCUMULATE" the total marks
		}						//End of the for loop for data entry
		printf("\nDONE\n\nFor the %i marks entered ….",numStudents);
		printf( "\nThe best mark is : %i %%",high);
		printf( "\nThe lowest mark is : %i %%",low);

		mean = total/numStudents;			//Calculate the average
		printf("\n\nThe average of these marks is %1.1f",mean);

	}							//End of the if for a good number of students
	else
	{
		printf("\n\nSorry, that is not an acceptable value.");
	}

	return 0;
}							//END OF THE PROGRAM!!
```

EXERCISES:

For each, ensure that you have used srand() to seed the random number generator.

1. Initialise an array of ten integers. Display the 10 numbers, the mean value, the highest value and the lowest value.

2. Input two integers to get the highest integer value and the lowest integer value from the user. Use the rand() function to fill an array of 100 integers which will be between these two inputted values. Output a table of these numbers and display the mean, low and high numbers.

3. Enter a single integer from the user between one and one hundred which will represent the number of integers to generate. Use the rand() function to generate this number of integers between 1 and 10 inclusive. Use an array called numOfEach that will store the count of the number of occurrences of each. (i.e. how many 1's, 2's etc. were generated)
   Output the results in an easily understood format.

4. Dice rolling simulation. The program will 'roll' two dice up to 1 000 times (number of times as selected by the user) and will count the number of occurrences of each sum of the two dice (totals between 2 and 12).
   Output the results as a bar graph or histogram.

5. Use the rand() function to generate an array of digits 1 to 9, each used only once. Output the original array and its inverse which is the array identifying the location of each digit.
   e.g.. original array 4 7 1 3 8 9 2 5 6
   inverse 3 7 4 1 8 9 2 5 6

   i.e. 1 is in the 3rd spot, 2 is in the 7th spot, 3 is in the 4th spot etc.